Measuring the Evolution of the Internet in the Age of Giants

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen University zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Master of Science

Jan Rüth

aus Hagen, Deutschland

Berichter:

Prof. Dr.-Ing. Klaus Wehrle
Prof. Bruce MacDowell Maggs, Ph.D.

Tag der mündlichen Prüfung: 14. 08. 2020

Abstract

The Internet has evolved into an essential cornerstone of modern life. At its core, it is seemingly still powered by protocols developed in the late 1980s. Since then, the Internet has experienced a colossal visible evolution, e.g., from delivering small text files over dynamic websites to highly interactive and bulky content that defines whole economic sectors. Notwithstanding, it is hard to believe that this change demanded no technological evolution, and in fact, research and industry have worked on many mechanisms and improvements to the core protocols. Today, we see Internet giants such as Google, Facebook, or Akamai controlling clients, servers, and networks driving these changes. Still, research has shown that these innovations often remain hard to deploy in practice as the Internet has condensed to supporting only a small set of protocols and parts of their features today.

In this dissertation, we design novel Internet measurement methodologies to gain an understanding of how the Internet has evolved from textbook knowledge, how it deviates from standardized practices, and how these discovered discrepancies affect Internet operation. To this end, we recognize the critical role of Internet giants and specifically investigate their impact on core Internet technologies.

From the bottom up, we start on the transport layer. Our analysis of TCP's initial congestion window shows that the Internet slowly converges towards recently standardized values. In contrast, Internet giants work outside these recommendations, and we find them configuring values that are several times larger. Moreover, we inspect the birth, deployment, and performance of QUIC, TCP's successor, finding that Internet giants have outdistanced the general Internet such that their market dominance challenges practical network operation. The silver lining is that Internet giants push their ideas towards standardization to be picked up at large. Nevertheless, auditing their use of congestion control shows that it is a core-differentiating property that crucially affects the bandwidth-sharing properties of today's Internet.

We utilize Internet control plane backscatter from our measurements to quantify the obsoletion in the Internet's core restricting its evolution. Apart from finding decade-long deprecated behavior, we uncover pathological cases of routing loops. Thus, we look up the stack and investigate how content owners utilize Meta-CDNs for traffic steering to innovate and liberate themselves from practice-dictating Internet giants. Our analyses show only a niche use of this technology, but it has the potential to affect CDN and ISP operation when applied at a massive scale.

Lastly, we explore the use of application layer innovations in large parts of the Web. JavaScript ubiquitously powers the interactive Web, and Internet giants push WebAssembly as its much more efficient successor. However, we discover that it is almost exclusively abused for browser-based cryptocurrency mining. Drilling down on this business, we expose that current blocking techniques are insufficient, leading us to design a novel method to estimate the generated revenues.

In summary, our contributions demonstrate that Internet giants coin practical Internet evolution today. While they are keen to standardize their innovations, their configurations are often a well-kept secret. Their market dominance challenges the innovation of others, but even they are not immune to an abuse of their technology.

Kurzfassung

Das Internet hat sich zu einem elementaren Eckpfeiler des modernen Lebens entwickelt. Den Lehrbüchern nach basiert es aber immer noch auf Protokollen aus den 80er Jahren. Von außen betrachtet hat das Internet sich jedoch enorm verändert: Von der Auslieferung kleiner Textdateien über dynamische Webseiten bis hin zu hochgradig interaktiven Inhalten auf denen ganze Wirtschaftszweige aufbauen. Natürlich war dieser Wandel nicht ohne neue Mechanismen und Weiterentwicklungen aus Wissenschaft und Industrie möglich. Diese werden heute maßgeblich von Internet-Giganten wie Google, Facebook oder Akamai, die Clients, Server und Netzwerke kontrollieren, vorangetrieben. Die Forschung hat aber gezeigt, dass es diese Innovationen oft nicht bis ins Internet schaffen, da nur eine kleine Anzahl von Protokollen und Teile deren Funktionalität in Wirklichkeit unterstützt werden.

In dieser Dissertation entwerfen wir neue Internet-Messmethoden, um zu verstehen, wie sich das Internet vom seinem Lehrbuchverhalten entfernt hat, wie es von standardisierten Praktiken abweicht und wie die aufgedeckten Diskrepanzen den Internetbetrieb beeinflussen. Dabei erkennen wir die kritische Rolle der Internet-Giganten und untersuchen gezielt deren Einfluss auf Kern-Internet Technologien.

Ausgehend von der Transportschicht betrachten wir zunächst TCPs initiales Staukontrollfenster. Wir zeigen, dass sich das Internet langsam in Richtung aktueller standardisierter Werte bewegt. Im Gegensatz dazu nutzen Internet-Giganten um ein vielfaches höhere Werte, als von der Standardisierung empfohlen. Außerdem untersuchen wir die Verbreitung von QUIC, TCPs Nachfolger, und vermessen dessen Leistung. Internet-Giganten haben den Rest des Internets abgehängt und wälzen durch ihre Marktmacht den Netzwerkbetrieb um. Jedoch treiben sie ihre Ideen zur Standardisierung und stellen sie der Allgemeinheit zur Verfügung. Bei genauer Überprüfung ihrer Staukontrolle zeigt sich indes, dass die Kombination aus Algorithmus und Parameterisierung die Effizienz und Fairness des Internets maßgeblich steuert.

Mithilfe von Reaktionen der Internet-Kontrollebene auf unsere Messungen quantifizieren wir zentrale Innovationshemmnisse. Abgesehen von jahrzehntelang veraltetem Verhalten, decken wir pathologische Fälle von Routing-Schleifen auf. Meta-CDNs umschiffen diese Hemmnisse über höhere Schichten. Unsere Analysen zeigen, dass ihre Verkehrssteuerung es Inhaltsbesitzern erlaubt Vorgaben von Internet-Giganten zu umgehen. Momentan fristen Meta-CDNs aber nur ein Nischendasein, hätten aber, bei großflächigem Einsatz, das Potenzial CDNs und ISPs merklich zu beeinflussen.

Schlussendlich untersuchen wir die Nutzung von Anwendungsschicht-Innovationen im Web. Dort ist JavaScript der Motor der Interaktivität, den Internet-Giganten durch das viel effizientere WebAssembly beerben wollen. Unsere Analysen zeigen aber, dass WebAssembly fast ausschließlich für die Generierung von Kryptogeld missbraucht wird. Wir brechen diese Praktik auf und stellen fest, dass aktuelle Sperrmechanismen nicht greifen, weswegen wir eine neue Methode zur Umsatzschätzung entwickeln.

In Summe zeigt unsere Forschung, dass Internet-Giganten die praktische Entwicklung des heutigen Internets prägen. Obwohl sie ihre Innovationen standardisieren, sind ihre Konfigurationen oft ein gut gehütetes Geheimnis. Ihre Marktbeherrschung stellt andere vor Probleme, aber auch ihre Technologien können missbraucht werden.

Acknowledgments

First and foremost, I want to express my gratitude towards Klaus, thank you for giving me a second home at COMSYS and allowing me to pursue my own research interests that eventually resulted in this book. Thank you for your advice and support over all these years. I also want to thank Bruce, not only for agreeing to serve as my secondary opponent but also for the numerous discussions at conferences and for just being the nicest person ever.

Further, I want to thank Ismet who supervised my Bachelor's thesis and inspired me to work as a student helper at COMSYS. Also, thank you, Hanno, for supervising my Master's thesis and subsequently encouraging me to pursue a Ph.D. Thank you for guiding and helping me at the beginning. Similarly, thank you, Florian, the discussions and your inspiration as my first office mate really helped me to get started and see problems from another angle. Without a doubt I am most grateful to Oliver; thank you for igniting my interest in Internet measurements, your advice, help and criticism, the numerous evenings before paper deadlines, and your guidance.

During my time at COMSYS, I had the pleasure of collaborating with many bright students which helped me to shape my work. Thank you, Christian, for your initial work on TCP, thank you, Pascal, for scaling it out, thank you, Ike, for pushing congestion control. I want to especially thank Konrad, thank you for your help with the Meta-CDNs, and browser-mining. You've really impressed me in your Master's thesis, especially with your dedication and efforts around the QUIC user studies.

I also want to thank all the awesome people at COMSYS. I want to specifically thank Hanno, Henrik, Torsten, Martin Serror, Helge, and Mike for being more than just colleagues. Thank you for all the heated discussions, papers, proofreading, conference trips, Kleinwalsertal excursions, and all the other non-work related activities. Thank you, Martin Henze for all your advice and discussions around browser-mining. Thank you, Constantin, for your relentless efforts in your Master's thesis and for being an awesome office mate. Thank you, Petra, Ulrike, Claudia, Kai, Dirk, and Rainer for keeping an eye out on the chair's work-life balance. Specifically, thank you Dirk for showing me how teaching is done, and thank you, Rainer, for showing me how a server room works and for keeping up with my endless scanning endeavors. To all the others that I did not mention above: thank you for making COMSYS a great place.

Last, but not least I want to thank my family and friends for their continuous support, love, and friendship. Thank you for shaping and accompanying me over the years. Above all, thank you, Lisa, for putting up with me, your distractions, understanding, patience, as well as the balance that you provide.

Declaration of Authorship

Parts of this thesis are based on the following peer-reviewed papers that have already been published. All my collaborators are among my co-authors. A detailed attribution of contributions can be found on the next pages.

List of Publications

[RBH17] Jan Rüth, Christian Bormann, and Oliver Hohlfeld. Large-Scale Scanning of TCP's Initial Window. In *Proceedings of the Internet Measurement Conference (IMC '17)*, pages 304–310. ACM, 2017. DOI: 10.1145/3131365.3131370.

[RPD+18] Jan Rüth, Ingmar Poese, Christoph Dietzel, and Oliver Hohlfeld. A First Look at QUIC in the Wild. In *Proceedings of the Conference on Passive and Active Measurement (PAM '18)*, pages 255–268. Springer, Cham, 2018. DOI: 10.1007/978-3-319-76481-8_19.

[HRW+18] Oliver Hohlfeld, Jan Rüth, Konrad Wolsing, and Torsten Zimmermann. Characterizing a Meta-CDN. In *Proceedings of the Conference on Passive and Active Measurement (PAM '18)*, pages 114–128. Springer, Cham, 2018. DOI: 10.1007/978-3-319-76481-8_9.

[RH18] Jan Rüth and Oliver Hohlfeld. Demystifying TCP Initial Window Configurations of Content Distribution Networks. In *Proceedings of the IFIP Network Traffic Measurement and Analysis Conference (TMA '18)*, pages 1–8. IEEE, 2018. DOI: 10.23919/TMA.2018.8506549.

[RZW+18] Jan Rüth, Torsten Zimmermann, Konrad Wolsing, and Oliver Hohlfeld. Digging into Browser-based Crypto Mining. In *Proceedings of the Internet Measurement Conference (IMC '18)*, pages 70–76. ACM, 2018. DOI: 10.1145/3278532.3278539.

[RKH19a] Jan Rüth, Ike Kunze, and Oliver Hohlfeld. TCP's Initial Window – Deployment in the Wild and its Impact on Performance. *Transactions on Network and Service Management (TNSM June '19)*, 16(2):389–402, IEEE, June 2019. DOI: 10.1109/TNSM.2019.2896335.

[RZH19] Jan Rüth, Torsten Zimmermann, and Oliver Hohlfeld. Hidden Treasures — Recycling Large-Scale Internet Measurements to Study the Internet's Control Plane. In *Proceedings of the Conference on Passive and Active Measurement (PAM '19)*, pages 51–67. Springer, Cham, 2019. DOI: 10.1007/978-3-030-15986-3_4.

[RKH19b] Jan Rüth, Ike Kunze, and Oliver Hohlfeld. An Empirical View on Content Provider Fairness. In *Proceedings of the IFIP Network Traffic Measurement and Analysis Conference (TMA '19)*, pages 1–8. IEEE, 2019. DOI: 10.23919/TMA.2019.8784684.

[WRW+19] Konrad Wolsing, Jan Rüth, Klaus Wehrle, and Oliver Hohlfeld. A Performance Perspective on Web Optimized Protocol Stacks: TCP+ TLS+HTTP/2 vs. QUIC. In *Proceedings of the Applied Networking Research Workshop (ANRW '19)*, pages 1–7. ACM, 2019. DOI: 10.1145/3340301.3341123.

[RWW+19] Jan Rüth, Konrad Wolsing, Klaus Wehrle, and Oliver Hohlfeld. Perceiving QUIC: Do Users Notice or Even Care? In *Proceedings of the International Conference on emerging Networking EXperiments and Technologies (CoNEXT '19)*, pages 1–7. ACM, 2019. DOI: 10.1145/3359989.3365416.

A Note on Previously Published and Joint Work

Parts of this dissertation are based on collaborations with students as well as with other researchers. The resulting publications form the scientific foundation of this thesis and were created with the support of the respective co-authors. We now attribute the different chapters and sections of this dissertation to the respective publications and authors. If not noted explicitly otherwise, the author of this dissertation was responsible for the initial concepts, methodologies, solutions, the implementations and evaluations, as well as the final publication.

- Christian Bormann laid the foundation for Section 3.1 in his Master thesis [Bor15]. The author of this dissertation designed the initial congestion window (IW) methodology, and Mr. Bormann implemented the first version. For the later publication [RBH17] a heavily reworked version of the implementation by this dissertation's author with help from Pascal Hein was used. The author of this dissertation performed the measurements, and he performed the analysis in collaboration with Oliver Hohlfeld. For Section 3.1.4, the author of this dissertation reimplemented the methodology to allow analyzing content delivery networks (CDNs) and together with Mr. Hohlfeld, analyzed the data and published the results in [RH18]. The dissertation's author performed the evaluation, and Alexander Löbel later validated it (through an independent implementation and evaluation) in his Bachelor's thesis [Löb18]. Ike Kunze implemented the author's design during his Master's thesis [Kun18], which is the basis for the performance perspective of IWs starting in Section 3.1.7.2.

Mr. Kunze, as well as Mr. Hohlfeld, helped in publishing parts of the results in [RKH19a].

- Our study on evolvable transports in Section 3.2 was designed and implemented by the dissertation author and analyzed with the help of Oliver Hohlfeld and appeared in parts [RPD+18]. Christoph Dietzel provided the Internet exchange point (IXP) data and helped in analyzing it. Similarly, Ingmar Poese provided the Internet service provider (ISP) data and its meta-information and also helped in analyzing the data. Konrad Wolsing helped in implementing and maintaining a framework used in the tool to capture QUIC connection parameters. Further, the foundation to Section 3.2.6 was laid in the Master's thesis by Konrad Wolsing [Wol19]. The technical evaluation appeared in [WRW+19] for which Mr. Wolsing performed all evaluations, the ideas and the methodologies were brought in by this dissertation's author. Further, the two user studies appeared in [RWW+19], also here, the studies themselves were supervised by Mr. Wolsing. He also contributed significantly to the design of the studies themselves.

- Section 3.3 on content provider (CP) fairness bases on the Master's thesis of Ike Kunze [Kun18] and parts of it appeared in [RKH19b]. This dissertation's author designed the methodology, and Mr. Kunze subsequently implemented and initially analyzed the data. Additionally, the author performed the later analysis in collaboration with Mr. Kunze and Mr. Hohlfeld that also appeared in [RKH19b] .

- Section 4.1 bases on the publication in [RZH19], it sources from measurements that are done in the context of Chapter 3 as well as of [ZRW+17, ZWH+18]. The software to process the data was designed and written by the author of this dissertation, Torsten Zimmermann helped in analyzing part of the data, especially the part about source quench (SQ) messages. Mr. Zimmermann, as well as Mr. Hohlfeld, aided in general discussions.

- Section 4.2 is the joint work of the author of this dissertation, Mr. Zimmermann, and Mr. Hohlfeld. Specifically, the parts relying on Ripe Atlas probes and the Cedexis Radar platform were the primary responsibility of Mr. Zimmermann. The author of this dissertation analyzed the Cedexis eco-system and developed the required tools and methods. Mr. Wolsing implemented the Raspberry PI-based measurement platform. Mr. Hohlfeld helped in discussing and integrating the findings into the context. The results of this section appeared in the joint publication in [HRW+18].

- Chapter 5 is based on the publication in [RZW+18]. This dissertation's author developed the methodology to find Wasm, and Mr. Wolsing implemented the initial prototype. The implementation finally used was done by the author of this dissertation. The methodology to associate blocks to a mining pool has its roots in private discussions with Martin Henze, who was not an author on the publication, and was subsequently developed from this idea by the author of this dissertation. Martin Coughlan from Symantec performed the actual

classification of websites to categories using Symantec's proprietary RuleSpace engine. Mr. Zimmermann and Mr. Hohlfeld helped in analyzing domain lists using the NoCoin filter list.

Contents

1

Introduction

The Internet is undoubtedly one of the most extensive and complex collaborations of systems that humankind ever built. Today, it has become a ubiquitous part of the everyday life that likely goes far beyond what the inventor's of early packet switching, as its foundation, had in mind. In this regard, an old Massachusetts Institute of Technology (MIT) computer handbook [Sta82] notes about the Advanced Research Projects Agency Network (ARPANET) (the early predecessor of the Internet),

> *"Sending electronic mail over the ARPAnet [sic] for commercial profit or political purposes is both antisocial and illegal."*
>
> — *Christopher C. Stacy [Sta82]*

In light of today's use of the Internet, such a statement seems ridiculous. Nonetheless, the ARPANET was a research network to explore the foundation of packet switching and networking. Only with its evolution to what we consider the Internet today, building on top of a universal addressing and a highly interconnected network of individual networks, the social and commercial interest of the individual participants grew (when realizing what possibilities it offered). These interests, of course, also resulted in a technological evolution of the Internet. To this end, Leonard Kleinrock, who is often credited as one of the Internet's pioneering inventors, wrote,

> *"In less than a year [after its first use on the ARPANET] email accounted for the majority of the network traffic."*
>
> — *Leonard Kleinrock [Kle10]*

Thus, already the early Internet showed that the availability of new technology drastically changes the way networks were used and utilized. Kleinrock further argues that *network measurements*, which he was responsible for, were a core principle in the ARPANET to understand what *they created, how people used it, and whether*

or not it was functioning as predicted or if models needed adaption. Still, these measurements were challenging, due to the high degree of cooperation that was required between the individual parties, and were eventually discontinued.

> *"In July 1975 responsibility for the ARPANET was given to DCA [Defense Communication Agency]. This terminated the systematic measurement, modeling, and stress testing that the UCLA [University of California] NMC [Network Measurement Center] had performed for almost six years, and was never again restored for the Internet."*
>
> — *Leonard Kleinrock [Kle10]*

Given the relatively small number of participating networks back then, such a systematic and comprehensive analysis of the Internet today, with its over 91 000 allocated autonomous systems (ASes) [Mai19], seems impossible just by the sheer numbers.

Internet Giants Transform the Internet

However, not all networks are equally important for the Internet as a whole, and recently, we have observed a flattening of the Internet's three-tiered architecture driven by large content providers (CPs) and Internet exchange points (IXPs) [GAL+08]. Today, end-users use the Internet and especially Web content with increasing access speeds [Aka16]. To this end, it was shown [EGR+11, TGD+18], similar to Kleinrock's initial observations about how technology drives networks, that today, videos are causing a substantial fraction of Internet traffic. These increasing demands to the network have led to a logical centralization of the *content-serving* Internet where a few big players serve the majority of the content [LIM+10, CSA+18]. These big players range from content delivery networks (CDNs), like Akamai or Cloudflare, over service providers, like Google or Facebook, to CPs themselves, like Netflix or Amazon. Today, they heavily peer across all network tiers, thus disrupting the traditional hierarchy of networks, and thereby, they practically reduce the number of involved networks between them and their customers (eyeballs). This reduced reliance on other networks helps in increasing the quality of the data delivery and in reducing transit costs but becomes only feasible once the overall traffic volume exceeds a certain level. Furthermore, many of these players are also heavily involved in not only serving the content but also in displaying them to the users. To this end, Google with its Chrome browser dominates browser market shares [Moz18] but also other companies such as Microsoft, Mozilla or Apple are heavily involved and shape terms and conditions how users perceive content. In summary, this reliance empowers all these *Internet giants*, and content owners are increasingly dependent on them, i.e., since they require the Internet giants' services (and are unable to realize them themselves), they also have to increasingly live at the giants' mercy and must adhere to their practices and pricing.

Still, these Internet giants enliven and challenge Internet evolution beyond a structural and economic level. Given their involvement in many parts of today's communications, they are in a pursuit for performance driven by increased revenues in response to

increased user satisfaction. At the Web Conference 2006, Marissa Mayer of Google reported that they lose 20% first-page searches when the search page needs twice as long to appear to a user [Lin06]. Amazon conducted similar experiments [Lin06]. They were able to show that sales dropped by 1% for an additional 100 ms page delay. Although the inverse, i.e., a 100 ms speedup will increase sales by 1%, is obviously not necessarily true, these numbers are often cited to justify the development of enhanced Internet protocols. To this end, we have witnessed the emergence of many new mechanisms, extensions, and protocols on the network, transport, and application layer that supposedly increase the performance. Still, it is largely unknown who actually uses these features and if they are practically beneficial, whether Internet giants themselves are applying them and if their use differs from what was actually specified. These questions become especially interesting since it is known that evolving the Internet is practically extremely challenging [Han06].

Internet Evolution is Practically Challenging

Enhancing existing protocols or devising new mechanisms is often not too challenging on a conceptual level. However, deploying these changes at large has become a significant hurdle. Back from the early days of the Internet, Kleinrock reports,

> *"[...] after a short grace period of a few months, no network was allowed to participate in the Internet if it did not comply with IPv4."*
>
> — *Leonard Kleinrock [Kle10]*

Today, the successor to the Internet Protocol Version 4 (IPv4), the Internet Protocol Version 6 (IPv6) whose Request for Comments (RFC) goes back to 1998 [RFC2460], is still in rollout, while deployment is on the rise [KGP+09, DLH+12, CAZ+14], it is still not ubiquitously supported 22 years later. These studies have a direct consequence for Internet content owners: if they wish to be reachable by the majority of Internet users, they need to acquire increasingly costly IPv4 addresses. Nevertheless, these deployment challenges have two roots. On the one hand, changes that require to update every connected system are impractical given the sheer number of connected devices. On the other hand, it turned out that even small changes that actually do not require cooperation are often impossible because implementors have *ossified* around the initial design [Han06] and made assumptions that the RFCs never stated.

These ossification challenges are especially apparent when looking at the network and transport layer. For example, it took over a decade [MAF05, TKB+15] to be sure that adding explicit congestion notification (ECN) to the Transmission Control Protocol (TCP) and the Internet Protocol (IP) would not harm connectivity. ECN allows routers to mark packets when they are congested rather than dropping the packet, which in turn allows TCP to reduce its rate without actually experiencing loss. One of the major concerns was that ECN uses two, previously unused, bits in the TCP header, which the RFCs initially reserved for precisely such extensions. Notwithstanding, given that this space was unused for decades, implementors assumed that the values of these bits must be zero and would drop the packet otherwise.

Thus, while using ECN gave clear performance advantages, it came with the risk of no connectivity at all, which did not help in promoting the feature.

These deployment challenges are not restricted to TCP's core header specifications. TCP even has a special option mechanism designed to extend it further. One of the most prominent examples is the story of Multipath TCP. Multipath TCP uses TCP's option mechanism to extend TCP to transport data over multiple paths through the Internet. Raiciu et al. [RPB+12] report on the issues and workarounds that they had to come up with to create a *deployable* Multipath TCP in the presence of an ossified Internet. They describe various kinds of *middleboxes* that break the Internet's end-to-end principle, such as firewalls, network address translations (NATs), or proxies, that demand extreme care when designing extensions such that connectivity never breaks. While they succeeded in building a deployable Multipath TCP, middleboxes still hinder its practical use, and thus, Multipath TCP must often fall back to regular TCP.

Takeaway. *The Internet giants' pursuit for performance and the difficulty to practically evolve core Internet technologies seem to be two mutually exclusive observations. Thus, we posit that, even though a comprehensive analysis of all participating systems in the Internet (as initially performed by Kleinrock) is impossible today, a focus on these Internet giants may allow a specialized but still globally applicable view on Internet evolution. Thereby enabling resuming to study how the Internet is used, whether it functions as assumed or if the current understanding in research and standardization needs updating.*

How Do Internet Giants Affect Internet Evolution?

In this dissertation, we draft behind Kleinrock's vision of practically understanding how the Internet evolves and if the evolved features can keep their promises with the help of network measurements. The Internet is an anarchic system without controlling entities that permit or forbid particular practices. Only standards govern the Internet operation; they, however, typically describe only a minimum for safe interoperability but leave room at the other end. By investigating how Internet reality diverges from standardization and textbook knowledge, we aim to update the understanding of Internet operation. This reality-check informs standardization on questions such as, whether standards are used in the way they were intended or if they need updating or whether Internet reality affects the operation of new standards. It further updates the understanding in research, e.g., fueling the development of new protocols by clearly painting practical challenges and realities that must be taken into consideration. Notwithstanding, we further hope to provide a lasting impact on education; teaching how the Internet operates from a purely academic point of view neglects the reality that will be faced post-education.

To provide this perspective on Internet evolution, we recognize the importance and power of Internet giants that aim to improve the Web and we design and perform Internet measurements to shine a light on their practices and the associated repercussions when accelerating the Web through transport layer optimizations. We further use Internet control plane feedback provided by these transport layer

measurements to analyze how initial assumptions baked into network protocols that have been reverted or that are now considered deprecated still prevail in the Internet. While our findings show a large portion of outdated systems, we continue to analyze how individuals can regain control of content routing in light of powerful Internet giants and an ossified Internet. Finally, we investigate how application layer optimizations that have been pushed by these Internet giants are (ab)used in the Internet and again emphasize Kleinrock's observation how technology *still* transforms the usage of networks today. Admittedly, designing Internet measurements to capture Internet evolution is challenging.

1.1 Research Questions and Challenges

To grasp an understanding of how the Internet has evolved, we introduce novel Internet measurement methodologies that are aligned to answer the following research questions.

- **What is the impact of Internet giants on Internet transport evolution?**

  With this question, we investigate how Internet giants evolve subject to the ossification on the transport layer. In this regard, we analyze how performance optimizations are applied in the Internet at large and by Internet giants in detail. How these optimizations relate to RFC-recommended practices as well as analyze the repercussions that may come with them. We further design methodologies that enable to investigate the birth of a new Internet transport and how Internet giants deploy it and thereby transform the Internet landscape at large. Our findings further motivated to investigate how the different performance optimizations practically affect the distributed resource sharing in the Internet.

  We thus paint a holistic picture of the current state of the Internet transport and thereby show its evolution.

- **How do content owners flexibilize in light of Internet giants and network ossification?**

  With this question, we shift the focus towards Internet players that are impacted by Internet giants and the non-existing evolution of the Internet core itself. To answer this question, we first take a look at how the network itself has ossified. We find that the public Internet is highly ossified, which renders network approaches infeasible for content owners that do not operate their own network. Thus, we look up the stack where evolution is still possible and practically analyze how content owners circumvent limitations put in place by Internet giants to allows them to flexibilize in light of giants and ossification.

- **How are application layer optimizations that are pushed forward by Internet giants used at large?**

  This final question further broadens the scope by investigating how people use the technology that Internet giants primarily developed and pushed towards Internet standards. Specifically, we look at how application layer technology that evolved rapidly over the past decades is now (ab)used in the Internet at large.

  By tackling this question, we provide a contrasting view to our earlier investigations.

We answer these questions by designing novel measurement methodologies that allow us to investigate the Internet at large and in detail. However, answering these questions involves tackling several challenges that we highlight in the respective chapters. We continue with an outline of the contributions of this dissertation.

1.2 Contributions and Outline of the Dissertation

We start our journey in discovering the evolution of the Internet from a historical viewpoint on the transport layer. Chapter 3 investigates how Internet giants strive for performance in an ossified protocol stack. Specifically, we first look at TCP, the de-facto transport since the 1980s and look at how TCP's initial congestion window (IW), as an end-host only parameter enabling tweaking the performance regardless of protocol ossification, has evolved. To do so, we develop measurement methodologies that enable us to investigate the IW distribution in all of IPv4's address space. We put a particular emphasis on how Internet giants use this parameter in speeding up the Web and find out that they operate way outside of current RFC-recommended practices. The chapter continues and looks at the current state-of-the-art in deployable transport protocols, namely QUIC. We were able to design measurements that monitor the birth and deployment of this new protocol. Further, we conduct human-centered evaluations of its performance and find that its superior performance is often not recognized by users. Still, our measurements highlight how Internet giants transform the Internet and that their market power is able to overwhelm operators. Motivated by our finding that congestion control (CC) often dominates protocol performance during our investigations of IW and QUIC performance, we close this chapter by looking at how Internet giants utilize CC. Our measurements show that Internet resources are not shared equally, especially the high traffic volumes of some Internet giants use CCs that dominate traditional defaults leading to highly asymmetric resource allocations.

Given the dominance of Internet giants today, Chapter 4 looks at how individuals can regain control of how they want to utilize the Internet infrastructure. To do so, we first investigate the state of the Internet's core in terms of deprecation and ossification around outdated standards and false assumptions. We develop a measurement methodology that sources from data that is created as a by-product from the measurements in Chapter 3 and shows that directly influencing the Internet's

core seems impossible given the large number of outdated systems. Given our observations, we turn our view up the stack where evolution has always been possible and investigate how Meta-CDNs utilize the Domain Name System (DNS) to perform policy-based rerouting of traffic between Internet giants. To this end, we dissect one of the most massive Meta-CDN deployments by Cedexis and investigate who uses it, for what purpose and whether or not it offers performance benefits for their users.

Finally, Chapter 5 completes our view on Internet evolution by inspecting how people use advancements on the application layer in the Web. However, we now focus on technology that has been designed by Internet giants not to work around Internet core limitations but to speed up the application layer itself. We will however not focus on how the Internet giants themselves use their innovations but rather how the rest of the Internet utilizes them. In this regards, we focus on WebAssembly (Wasm) as the new efficient alternative to JavaScript that ubiquitously powers the interactive Web. Driven by media reports on browser-based cryptomining, we find that cryptominers often utilize Wasm due to its high efficiency. We thus design a measurement methodology to specifically investigate the use of Wasm on large parts of the Web. Our measurements show that people use Wasm already today. However, we were able to attribute over 96% of all Wasm code to miners by developing a novel fingerprinting methodology that outperforms current blacklist approaches. Given that people currently abuse Wasm for mining the Monero currency, we set out to dissect the largest browser-based cryptomining provider Coinhive. To this end, we develop a novel methodology that enables us to associate blocks in Monero's privacy-preserving blockchain to a mining pool and specifically Coinhive. This methodology enables us to statistically investigate Coinhive's user base as well as the revenue generated through browser-based cryptomining.

Before we dive into this journey, Chapter 2 provides the necessary technical background knowledge to understand the design decisions that we took in our Internet measurements. First, we highlight how the Internet's architecture and specifically how Internet giants have changed it over the past decades. In that regard, we investigate DNS as a core technology used by CDNs for content routing. Furthermore, we highlight how the protocol stack has ossified around a small number of protocols and false assumptions about them. We continue by showing the general approach to Internet measurements as the foundation for our investigations. Subsequently, we highlight several core technologies investigated in this thesis, i.e., TCP's CC with its fairness properties and active queue management (AQM) as a solution to many of TCP's problems, as well as QUIC as the foundation of the Hypertext Transfer Protocol (HTTP)/3 and TCP's likely successor for the Web.

2

Background

This chapter provides the necessary background information on Internet technology and wraps up crucial keystones in its evolution that help us in understanding why we focus on particular technologies, make design choices, and come to conclusions. It is not meant to replace any lecture on data communications even though, in parts, we will reiterate on some core concepts.

2.1 Internet Architecture and the Rise of Internet Giants

We first dive into the structure of the Internet as a foundation for nearly all our later analyses that regard network ownership or routing. After this brief structural introduction, we shift the focus to the Internet protocol stack and its historical ossification to understand why Internet giants pursue the technological shifts we analyze in Chapter 3.

Internet Architecture

The Internet is a network of individual networks, the autonomous systems (ASes); in May 2019, there were roughly 91 000 allocated ASes [Mai19]. Each AS is uniquely identified using an AS number (ASN). These numbers and also Internet Protocol (IP) addresses are assigned to the ASes through the Internet Corporation for Assigned Names and Numbers (ICANN) via regional Internet registries (RIRs).

According to [RFC4271] an AS is a collection of several routers under a single *technical* administration that uses an Interior Gateway Protocol (IGP) to route packets within the collection of routers according to a set of metrics. For interconnect, the different

ASes use the Boarder Gateway Protocol (BGP) [RFC4271]. In essence, if two ASes decide to *peer*, i.e., connect each other, or one buys *transit*, i.e., network access, from the other, they utilize BGP to exchange reachability information in the form of *prefixes*, i.e., IP address ranges, and AS-paths to reach these prefixes. When an AS decides to transport data for another AS through its own network (transit) to reach an IP address, it will forward its learned prefix/AS-path pairs to the other AS adding itself to the front of the AS-path. This transitive exchange of information eventually forms the basis for *routing* decisions in the Internet.

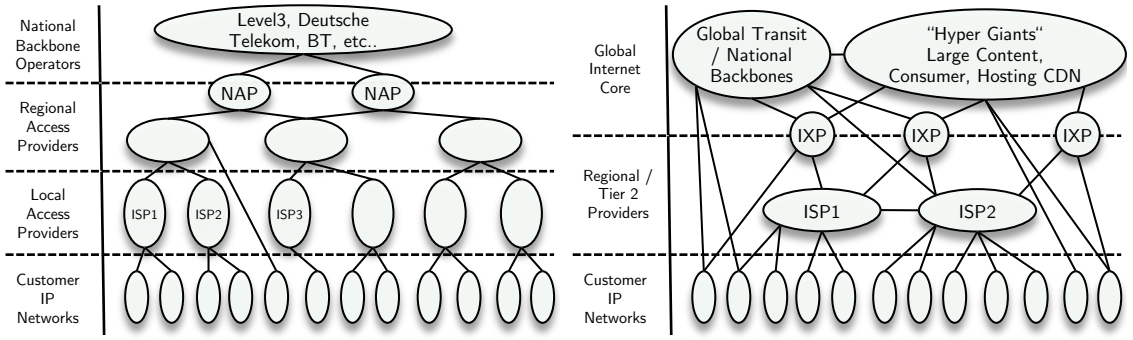
Each AS uses this information to calculate the "best" next-hop router that should forward a packet. It is important to note that "best" does not always mean optimal in terms of performance. BGP allows defining several metrics that can be used by the network administrator to prefer one route over another (e.g., there can be multiple peers that enable reaching the same destination). Typically, when two networks peer, they enter a business contract that explicitly defines service level agreements (SLAs) that govern what kind of traffic both parties may exchange. In that regard, BGP is merely a framework that enables to reflect these contracts on a technical level. Thus, given multiple peers to reach a destination, a network administrator can utilize BGP to perform traffic engineering (TE) to (cost) efficiently route outgoing traffic or influence routing decisions of its peers, e.g., given specific expected traffic volumes. This TE also implies that a packet does not always follow the AS-path seen at the source as all intermediate ASes perform individual routing decisions. Furthermore, it also means that an answer to a packet does not necessarily follow the same route back to the original sender, which is generally known as Internet path *asymmetry*.

Autonomous System Peering Structure

In the early Advanced Research Projects Agency Network (ARPANET), universities and some military corporations were interconnected [Kle10] *logically* to experiment with packet switching. We emphasize logically since, of course, these different institutions were spread over the North American continent, and a *physical* connection was only possible with the help of large telecommunication providers that operated the telephone network. Thus, physical telephone lines were rented to interconnect the institutions.

This physical reliance on large telecommunication providers has also led to the traditional tiered structure of the Internet found in every textbook which we depict in Figure 2.1a. They provided the backbone to which several regional providers connect, in turn offering connectivity to Internet service providers (ISPs) which offer access to regular customers/users.

Recently, Gill et al. [GAL+08] found that this hierarchy started to flatten. They found that the emergence of content delivery networks (CDNs) and other large content providers (CPs), as well as the proliferation of Internet exchange points (IXPs), disrupted this architecture [LIM+10] in favor of what we depict in Figure 2.1b. As these CPs had to rely on the large Tier-1 providers to disseminate their content to the users, they also required to buy transit, which, given their significant amount of outgoing traffic, was a costly matter, even though transit prices have been falling



(a) The traditional, textbook, Internet architecture.

(b) The recent Internet Architecture that is impacted by highly connected Internet Giants and IXPs.

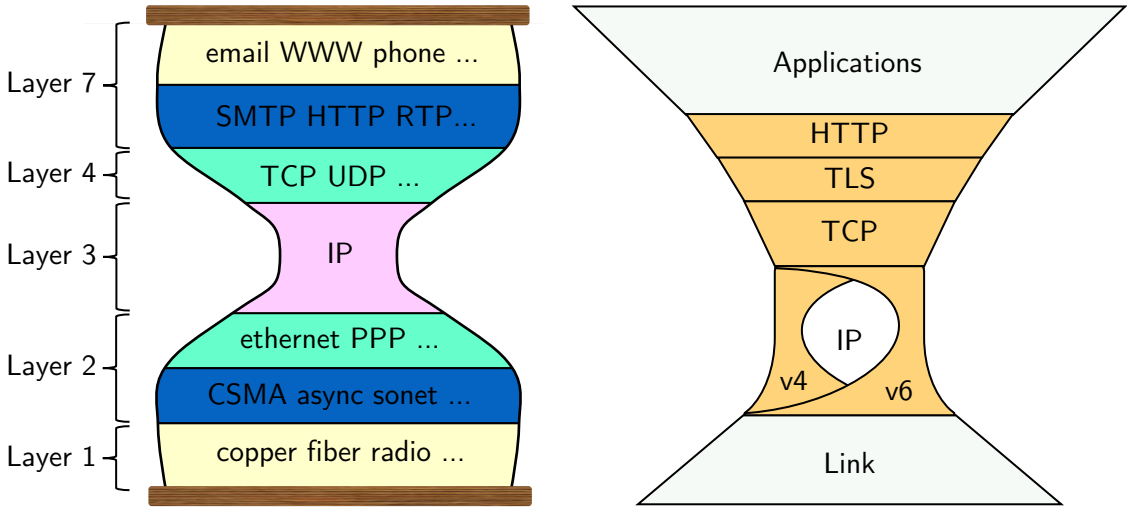
Figure 2.1 The traditional and more recent Internet architecture. Adapted from [LIM+10].

since the end of the last century (by about 30% each year [Nor14]). Still, prices are known to be a matter of negotiation, commitment, and many other factors which makes planning difficult [VLF+11]. IXPs, to this end, usually offer transparent pricing where one buys a link with a specific capacity and then the IXP enables one to reach all other networks that also are peering at the IXP. While large Tier-1 networks still operate the global backbone, the Internet giants flatten the traditional hierarchy by heavily peering across all tiers often with the help of IXPs. Thus, many Internet giants do not critically rely on other networks to reach their customers, putting them into the position to rapidly evolve.

Internet Protocol Stack

Nevertheless, also Internet giants are bound to the technological structures that the pioneering inventors of packet switching set in place in the early 1980s. Still, from the early beginnings, they designed the Internet for evolution. To enable this evolution, IP, or more specifically back then IP Version 4 (IPv4), was mandated as a common routing and addressing scheme. Depicted in Figure 2.2a, this ubiquitous availability forms the "narrow waist" of the Internet that enables evolution below and above the IP layer. Indeed, this reliance on IP enabled a plethora of access technologies below, as well as, many transport and application layer protocols above.

However practically, especially above IP, development and deployment of new protocols did not go hand in hand. This struggle had several reasons. Two of them being; first, protocols were often developed in academic content, and commercial Interest was limited, which rendered it difficult to get others to deploy the protocols. Second, the Internet had started to *ossify* around a couple, heavily used protocols, especially on the transport layer. For example, since the Internet could not sustain the growth, and it was apparent that Internet addresses were a scarce resource in IPv4, network address translation (NAT) was deployed to fight address exhaustion. Aggravatingly, these *middleboxes* started to alter IP and transport header information to fulfill their task, thereby violating the end-to-end principle. While middleboxes are debatable, their use is often necessary for network operation. However, they critically challenge



(a) The Internet protocol hourglass. Adapted from [Dee98].

(b) The dual-twisted or two-stemmed ossified Internet "martini glass". Adapted from [TH14].

Figure 2.2 The traditional and new (ossified) Internet protocol stack.

the deployability of new Internet protocols. To continue the NAT example, NATs typically only implement address translation for the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP), the two most common transport protocols, and drop every other transport. This constraint critically hinders the deployment of new protocols, yet it further turned out that many boxes make additional assumptions on the Internet protocols. It was shown (e.g., [RPB+12]) that standard-compliant extensions to TCP or IP are not deployable due to these boxes.

Today, the Internet protocol stack is thus widely regarded as depicted in Figure 2.2b. IP is still the common substrate; however today, IPv4 co-exists with IP Version 6 (IPv6) that is still in rollout. Above IP, most applications make use of TCP and often Transport Layer Security (TLS) for security, and due to its simplicity and success in the Web, use some version of the Hypertext Transfer Protocol (HTTP). Even applications that could use a better architecture, e.g., building on UDP, still utilize this architecture such as Youtube for video delivery. This success has also lead academia and industry to focus on HTTP and TLS for innovation, however, as we will see in the later chapters, TCP poses insurmountable hurdles which are currently pushed by Internet giants.

We continue by looking at the Domain Name System as it is the foundation for human-rememberable addressing and is also heavily utilized by many large CPs for routing users to close-by content.

2.1.1 The Domain Name System

Early on in the ARPANET, it became evident that remembering numerical addresses is a challenging task for humans. To this end, already in 1973, it was suggested [RFC606] to have a central repository containing an authoritative file

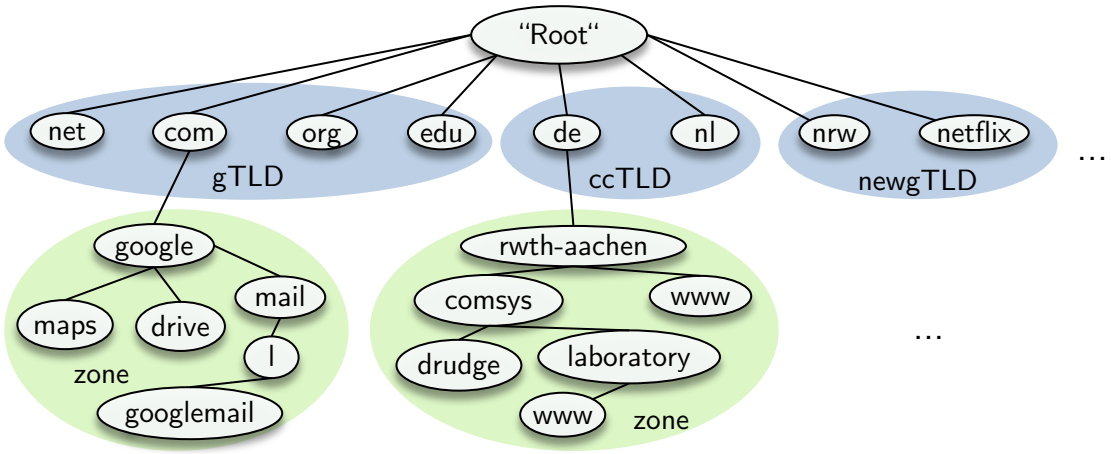


Figure 2.3 Logical DNS tree structure. Leaves denote hosts whose full DNS name is the concatenation of labels from the root to the leaf. Different portions of the tree are managed by different name servers in several zones.

mapping addresses to human-readable hostnames. In the 1980s, a single person was responsible for this file [RFC882]. Thus, holidays, as well as the required update frequency due to the Internet's growth, made it hard to manage.

These and other problems gave birth to the Domain Name System (DNS), a hierarchical tree-like logical segmentation of the domain name space that allows distributing the management to many *authoritative name servers*. However, DNS is more than a logical concept. It is also a protocol that defines how *resolvers* can retrieve and lookup information in this highly distributed database.

Figure 2.3 shows a small, incomplete portion of the DNS tree. All top-level domains (TLDs) are reachable via the root zone; among others, there are generic TLDs, country-code TLDs, and more recently new generic TLDs, all are managed by ICANN. One or more name servers (NSes) under a single administrative domain manage each zone and are in turn referenced in their hierarchically preceding zone. A DNS hostname is the concatenation of each label from the leaf to the root separated by a period, where the root zone's label is empty.

The root zone itself is made up of 13 logical root servers that are managed by 12 independent organizations, labeled a, b, c, ..., to m. Each server is however replicated for fault tolerance; to this end, root servers make heavy use of anycast routing.

To enable users to resolve a hostname, e.g., to an IP address, a resolver is required. It is the resolver's task to iteratively query the DNS tree until it has reached a leaf. This resolution involves a series of message exchanges, at first, when starting from scratch, the resolver will contact one of the root servers (operating systems typically provide a hints file containing some root server addresses). The resolver will then request resolution, e.g., for an A-record (an IPv4 address), of a hostname. Since it is outside of the root server's jurisdiction, it will try to point to a name server (using an NS-record) that has more information. Taking the example of `www.laboratory.comsys.rwth-aachen.de`, it will point to the name server of `.de`, one of them is named `a.nic.de`. Since we require an IP address, a subsequent resolution of this record is required. For reasons of efficiency, a server typically

```
$ dig @134.130.4.9 +norecurse www.laboratory.comsys.rwth-aachen.de

;; ANSWER SECTION:
www.laboratory.comsys.rwth-aachen.de.   172800 IN   CNAME    laboratory.
    comsys.rwth-aachen.de.
laboratory.comsys.rwth-aachen.de.       172800 IN   A       137.226.59.41
```

Figure 2.4 Output of the `dig` command line tool (reduced to the essential part) for the final DNS resolution step of the example. Each answer contains the requested name, a TTL, a class, the record type and the answer.

provides this record directly together with the NS-record in an additional section of the DNS answer. Now, the resolver can contact the authoritative DNS server for the `.de` zone, and it can again ask for the A-record of the hostname that we want to resolve. However, the server again redirects us to `rwth-aachen`'s name server with the same mechanism as before. Finally, when contacting this server, it is able to answer the query as it is the authoritative name server for this zone. Even though there are additional subdomain parts in the hostname, it is not clear before resolving how zones are divided. In case of our original query for an A-record for `www.laboratory.comsys.rwth-aachen.de`, the server will answer with two records as shown in Figure 2.4.

The first record contains a canonical name (CNAME)-record, i.e., a record that maps one name to another. Usually, one would resolve this record similarly to every other record. In this case, `rwth-aachen`'s zone is also responsible for the redirect's target. Its DNS server directly provided the A-record as well.

To speed up this tedious resolution process, DNS resolvers can cache records according to each record's time to live (TTL). In our example, both records have a lifetime of 172 800 s, thus, are allowed to be cached and do not need to be resolved again for 48 h. To draft behind this feature, DNS has the concept of a *recursive resolver*. Instead of having every single user perform the aforementioned iterative queries, a central resolver, e.g., operated by the user's ISP performs the iterative queries. To this end, each user forwards her query to the central resolver asking for recursive resolution. After the resolver got the final answer, it will forward it to the original requester. Thus, several requests from different users can effectively utilize the caching that resulted from a *single* user's resolution.

While DNS offers many other concepts, for this dissertation, we will mostly make use of CNAME and A-records. Notwithstanding, CDNs heavily use both concepts as we will see in the following.

2.1.2 Content Delivery Networks

In the early 2000s, operating websites became increasingly complex as the user base became more and more international, the complexity of operating a Web

server and optimizing the website itself was steadily rising, and most importantly it became increasingly complex to scale to the vast number of users. At the same time, companies found that unresponsive and long-loading websites harmed users' consume-behavior. To this end, CDNs came to the market to solve the problems mentioned above. In essence, one can regard CDNs as a globally distributed website cache. By replicating website content around the world, users can be directed to nearby content caches to be served quicker and more efficiently as CDNs massively optimize websites as well as their delivery. Nevertheless, not all content is cacheable, e.g., when a user logs in to a password protected part of a website or if a website shows user-specific content. To this end, as well as when content is cacheable but not in the cache, the CDN's server then forwards these requests to the *origin* server, i.e., the original Web server, to be answered and relays the answer as if it produced it itself.

To serve users from a nearby cache, a CDN thus needs to direct users to such a cache. Two common ways are using anycast or DNS. When using anycast, CDNs try to optimize the routing to be able to direct users to a close-by cache. We will focus on DNS in the following as it is still vital also for anycast-based CDNs.

DNS for User to Content Routing

As the DNS is responsible for resolving names to IP addresses (see Section 2.1.1), a CDN must decide on a target server that should serve the content during the DNS resolution. To this end, the resolver must contact the CDN's NS. However, the CDN's NS is typically not the authoritative NS of a domain and is thus not contacted during the DNS resolution.

Let us take the example of `www.tagesschau.de`, the news website of the German public-service television which is hosted with the help of the Akamai CDN (one of the largest CDN operators). When we follow the DNS tree for `www.tagesschau.de`, the `.de` name server redirects us to `ns1.dunkel.de` which in fact is one of the authoritative name servers. However, instead of providing us with an A-record, it uses a CNAME-record to effectively redirect us to a different domain name, in this example, `san.tagesschau.de.edgekey.net`. As we can see, this hostname is very different in that it is not only hosted in a different TLD but also does not share the same domain. Thus, effectively, the CNAME-record was used to change the authoritative NS. In this case, Akamai operates `edgekey.net`, and this way, `www.tagesschau.de` delegates control to Akamai. As our DNS resolver will now continue to resolve the CNAME, Akamai is now contacted and thus enabled to resolve to a server of Akamai's choice. While this example was taken from Akamai, it is typical for a DNS-based CDN. Two key observations can be made; first, typically control is handed over to the CDN via a CNAME-record. Second, the CNAME record effectively encodes the original request such that the CDN knows which content a user is asking for. This information is then used to select a suitable content serving server.

Typically, a CDN optimizes this choice with regard to latency, available bandwidth, server load, availability of caches, and operational costs. While the exact mapping

```
$ dig @8.8.8.8 www.tagesschau.de

;; ANSWER SECTION:
www.tagesschau.de.  173 IN  CNAME    san.tagesschau.de.edgekey.net.
san.tagesschau.de.edgekey.net. 16555 IN  CNAME   e8178.e6.akamaiedge.net.
e8178.e6.akamaiedge.net. 19 IN  A   184.31.89.157
```

Figure 2.5 Output of the `dig` command line tool (reduced to the essential part) when requesting recursive resolution at 8.8.8.8 (Google's public DNS server) for `www.tagesschau.de`.

function is a business secret, CDNs today usually try to minimize latency towards their customers for increased interactivity and faster delivery. Latency in the Internet mostly depends on two factors. First, the physical length of the medium (e.g., cables or wireless links) between CDN and user, and second, the queuing delay within each router on the path. Achieving low latency is challenging as the CDN is unaware of the user's address as a recursive DNS resolver usually performs the DNS resolution on behalf of the user. CDNs nevertheless assume that users are in close proximity to the recursive DNS resolver, a reasonable assumption, as ISPs and other DNS operators in turn also try to minimize the latency between users and DNS servers for increased DNS responsiveness. Thus, CDNs periodically determine the round-trip time (RTT) between their servers and the DNS resolvers, e.g., using network pings, building a latency matrix. With the help of this matrix, as well as the other metrics, a CDN's DNS server can now perform an informed decision to which content serving server it best redirects a user.

In Akamai's case, this decision step happens twice, which we depict in Figure 2.5 using `dig`'s output. The first line of the answer section shows the CNAME-redirect that we discussed earlier, the DNS server demands a similar redirect in the subsequent line. In fact, Akamai operates global DNS instances which redirect us to a local DNS instance that has more precise knowledge (redirected to in the second answer line). One critical observation that one can make is that the TTLs are vastly different between the different records (173 s, 16 555 s, and just 19 s). Since a DNS recursor will cache the records, it will answer subsequent requests to the same hostname from its cache. Thus setting large values reduces the DNS-load on the CDN's DNS servers and users profit from the caching itself, but on the other hand, the CDN's DNS server is unable to redirect users somewhere else during the record's lifetime. It is just a tradeoff that CDNs have to perform between being able to quickly steer users, making use of caching, and managing their servers' load.

Today, CDNs have become an integral part of the Internet. Their services go far beyond simple caching; they are used to protect websites against attacks, they optimize a website structurally and, as we will see in Chapter 3, they profoundly optimize their transport protocols. However, there are also more advanced concepts that we investigate in Chapter 4 and that are currently challenging CDN operation and their profits. Additional information regarding CDN operation and their design space beyond what we presented here can be found in [DMP+02]. We continue with a general discussion of Internet measurements before looking at transport protocols.

2.2 Internet Measurements

Internet measurements are the foundation on which this dissertation builds its methodologies. Generally, one distinguishes between *active* and *passive* measurements, which we employ both. Each approach has advantages and disadvantages, and it significantly depends on the measurement goal which of both is more suitable.

Active Measurements

In active measurements, as the name suggests, one actively participates in the communication. This partaking can range from setting up connections to injecting some other kind of probe traffic to stimulate a specific response that one can then observe. This full control over the measurement is one of the advantages of active measurements. To this end, the operator can study specific system or network properties in situations that would otherwise rarely or never occur, e.g., to perform penetration testing or discover worst-case performance. However, on the downside, active measurements interact with live systems, thus also affecting their operation and potentially the measurement itself. Furthermore, active measurements typically do not allow to gather insights about usage patterns, e.g., actively measuring the RTT to every Web server in the Internet may yield interesting insights, however, it does not easily give insights into which latency website users actually experience when visiting the respective Web servers. Active measurements are, to this end, also *vantage point*, i.e., measurement location, specific and it is typically hard to gather representative vantage points as we will also see later in Chapter 3 and Chapter 4.

Passive Measurements

Passive measurements, on the other hand, do not inject any traffic but only observe traffic through traffic captures. These captures can range from full traffic traces to sampled traffic. The main advantage of passive measurements is that they enable to gather "real" traffic and thereby behavior. Thus, they also do not interfere with the systems under investigation. Similar to active measurements, the availability and choice of vantage points also challenge passive measurements. It is typically tough to get access to these vantage points as there are significant privacy and operational concerns. Additionally, when designing passive measurements, it must be taken into consideration that the choice of vantage point significantly biases the study outcome. For example, what is the user population that one observes, i.e., does one observe actual users, machine-to-machine communication, or a mixture? At what time of day is the study performed? So, passively measuring, e.g., the RTT to Web servers at night could paint a drastically different picture than during peak hours (e.g., due to more congested links).

Whatever the choice in methodology, Internet measurements should generally follow basic ethical guidelines which we present in the following.

2.2.1 Measurement Ethics

Since Internet measurements affect many entities, it is crucial to assess the consequences and implications of the measurements onto the individual parties. These range from technical, e.g., how fast should packets be generated, to social, e.g., which information should be stored, questions. One of the critical problems in Internet measurements is that it is usually impossible to ask the involved parties for consent; e.g., when scanning all of IPv4 for Web servers, it is impossible to contact each and every operator upfront. Thus, Internet measurements should generally not affect the operation, management, and costs of Internet services or networks, i.e., measurements, especially active ones, should not exploit security vulnerabilities or crash the involved systems. Furthermore, the privacy of user data should be guaranteed, so it is common practice to either remove all privacy-related data or to scramble it beyond reconstruction. In addition, since upfront consent is not possible, one should provide opt-out mechanisms for operators, especially in the case of active measurements. To this end, operators should be able to identify the nature and origin of the measurement traffic effortlessly.

Durumeric et al. [DWH13] recommend seven practices for "good Internet citizenship" when doing large-scale active Internet measurements:

- **Close coordination with the local network administrators.** This allows assessing the risks of measurements by sourcing from the operational experience of the local operator. It enables accounting for potential operator-side biases, e.g., there might be firewalls or Web caches that can affect the measurement itself. Further, it allows establishing means of handling inquiries.

- **Verify that measurements do not overwhelm the network.** This is to make sure that the local, as well as the upstream provider, can handle the traffic and that it does not affect the regular network operation.

- **Signaling the benign nature.** The machines from which the traffic originate should provide websites that clearly state the purpose of the measurements. Furthermore, reverse DNS (rDNS) entries should be used to hint at the benign nature, e.g., looking up one of our measurement machines `137.226.113.8` yields `researchscan1.comsys.rwth-aachen.de` as the hostname signaling its use as a research machine. The measurement itself can provide further information, e.g., the HTTP user-agent header that Web servers often report in their logs can contain a website with additional information.

- **Be precise about the measurements.** In all communications, the scope and purpose of the measurements should be clearly communicated.

- **Provide simple means to opt-out.** It should be easy for network operators to opt-out from the measurements, and one should handle opt-out requests promptly. For example, one can provide an email address with a template stating the required information to opt-out of one of the measurement websites.

- **Conduct measurements no larger or frequent than required.** The measurements should accurately focus on the research objective to keep the footprint low.
- **Spread measurement traffic**. Measurement traffic can be spread over time and over multiple source addresses. While this partly contradicts with being easily detectable and upfront, it is nevertheless a good idea to not overwhelm networks with the measurement traffic.

In the measurements on which this dissertation builds, we have followed all of these guidelines. In the following, we discuss the fundamental technologies that our measurements target, starting with transport protocols.

2.3 Internet Transport

Transport protocols are a fundamental building block in the communication stack. Traditionally, network engineers have implemented them in the operating system offering different service primitives to applications. For this dissertation, we focus on protocols that enable a reliable communication, i.e., protocols that are able to deliver cohesive data in the order in which they were sent (in-order) and recover from losses or transmission failures (fault tolerance). Reliable transport protocols are necessary as IP only offers a best-effort hop-by-hop data delivery, i.e., IP does not guarantee any data delivery. Nevertheless, many Layer 2 protocols are concerned with reliable point-to-point transmission (but they are rarely used), i.e., they recover from transmission errors. Still, a network building on reliable Layer 2 technology may be subject to packet loss.

This susceptibility to packet loss is rooted in asynchronous nature in which networks are operated. Today, each router that handles a packet must implement some form of queues. First, to store incoming packets and then, after deciding how to forward them, proceed to enqueue them in an egress queue that corresponds to the forwarding decision. These queues already foreshadow the problem. Imagine a router with three 1 Gbit/s interfaces. Now traffic from two of these interfaces must be forwarded to the third interface. This forwarding poses no significant problem as long as the traffic of both ingress interfaces does not exceed the data rate of the egress interfaces. However, if now both ingress interfaces receive traffic at 1 Gbit/s, 2 Gbit/s must be transmitted over a link with only 1 Gbit/s data rate which is obviously not possible (regardless of the available queue size). Thus, data in the egress queue will accumulate until the queue is full, and, ultimately, force the router to drop packets. Increasing the queue size solves the problem temporarily but eventually leads to the same situation and offers no viable solution. Thus, even if networks were to use a reliable Layer 2 transmission, packets can be lost in Internet communication, and thus transport protocols must handle these kinds of failures.

It is thus also a key concern of transport protocols to manage this *congestion*, i.e., to dynamically react to changes in the network, which is generally known as congestion

control (CC) that we discuss in more depth in Section 2.3.3. Today, networks have queues to enable capturing bursts and handling temporary mismatches of data rates such that the network utilization can be kept high. However, queues induce latency, i.e., a packet must wait until all preceding packets are transmitted, and network operators thus face the challenge of keeping a high utilization (which would be offered by a big queue due to having many packets available for transmission) while having fast transmissions through low queueing delays (which would be offered by small queues due to short to no waiting times). To this end, research has worked on router buffer sizing to investigate appropriate sizes for queues as well as on how to manage queues efficiently going beyond simple *drop-tail* queues, which we discuss in Section 2.3.4. As it turns out, the problem is not solvable in the routers alone and requires an interplay between end-hosts implementing CC and the router queues.

We continue by describing the fundamentals of TCP as the de-facto standard transport protocol of the Internet since the 1980s and QUIC, the current candidate to succeed TCP for the Web and potentially other application areas.

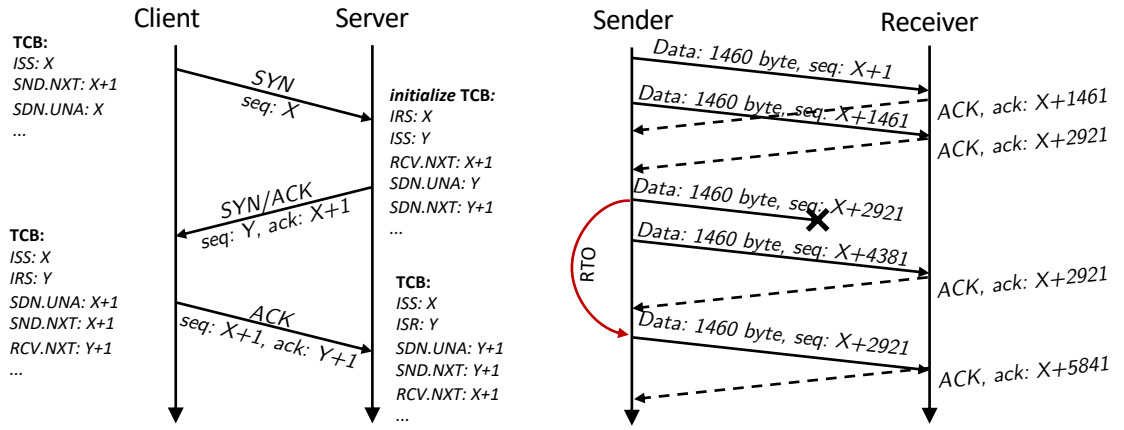
2.3.1 The Transmission Control Protocol

The Transmission Control Protocol (TCP) was developed in the context of the ARPANET, and one can find its initial specification in [RFC793] from 1981. Today, over eight Requests for Comments (RFCs) specify its core functionality, 26 additional RFCs exist that specify strongly encouraged functionality, and there are 20 further RFCs specifying experimental extensions. Thus, we will refrain from providing a holistic explanation of TCP, but we are going to refresh the bare minimum that allows comprehending our TCP-based measurements in Chapter 3 and their motivation. For further reading, we recommend the current effort [Edd19] of uniting the eight core-documents to a single specification that is supposed to become RFC 793bis.

TCP provides the bidirectional delivery of a reliable in-order byte stream between two endpoints. To this end, TCP splits the data into several segments that are in turn delivered in individual packets and on reception recombined in the right order and handed over to the application process that is using TCP. TCP enables its service through a stateful connection and meta-data that it transmits in the header of every segment. During connection establishment, both parties exchange vital information that allows them to derive a mutual connection state, which is the foundation for data exchange and loss recovery.

Connection Establishment

TCP uses a three-way handshake, depicted in Figure 2.6a, to establish a pair of sequence numbers for data enumeration and for data acknowledgment, which TCP stores with the help of a transmission control block (TCB) in main memory. TCP updates the TCB throughout the lifetime of the connection to, e.g., keep track of unacknowledged bytes (UNA) or which bytes it should send (SND) or receive (RCV) next (NXT). It uses a special header flag (SYN) to signal the desire to establish



(a) TCP's 3-way handshake and resource allocation.

(b) Simple data exchange in TCP with transmission failure and retransmission timeout (RTO) segment recovery.

Figure 2.6 TCP connection establishment and data exchange.

a connection. To signal correct SYN reception, each TCP endpoint acknowledges the receipt of the other parties sequence number by mirroring the received sequence number while increasing it by one.[1] Thus, after the first segment, the receiver will answer with its own sequence number while mirroring the received one adding 1 to it (thereby acknowledging the receipt and indicating the next sequence number) and also setting the SYN flag. Finally, TCP again acknowledges the correct receipt, yet, as both parties already exchanged the sequence numbers, this segment does not carry the SYN flag. Now both hosts have successfully exchanged sequence numbers from which they can now commence to relatively address bytes in the byte stream. During the handshake, TCP further allows establishing additional optional state, e.g., which maximum segment size (MSS) it likes the other end to use.

Data Exchange

To exchange data, TCP splits the stream into (ideally MSS-sized) segments. Each segment carries one sequence number that reflects this segment's first byte in the overall byte stream (relative to the initially exchanged numbers). This numbering allows the receiver to bring received bytes into the right order as well as to acknowledge their correct receipt. To this end, TCP acknowledges the last byte that it received and was able to bring into gapless order by announcing the byte it expects next, as shown in Figure 2.6b. In a widespread extension, TCP uses selective acknowledgmentss (SACKs) to signal the reception of out-of-order segments allowing for a more efficient retransmission mechanism. Notwithstanding, when TCP does not receive an acknowledgment for its data after a specific time, the retransmission timeout (RTO), TCP regards this segment as lost and will retransmit

[1] TCP actually uses the sequence number to enumerate the bytes in the byte stream; however, there are instances where it is additionally used to signal correct reception of special segments such as here in the handshake.

it (again depicted in Figure 2.6b). In the so-called *fast retransmit* mode (not shown), a TCP sender uses the reception of three duplicated acknowledgments (ACKs) as an indicator that a segment was lost (e.g., due to a transmission error or a full queue). Imagine a single packet loss in the middle of a train of ten packets. The subsequent packets will be received, and TCP will send acknowledgments (useful especially with SACKs). However, given the gap, TCP will acknowledge only up to the gap (also shown in Figure 2.6b at the third ACK). The cumulative acknowledging thus results in multiple duplicated ACKs which fast retransmit uses as an indicator for loss which is significantly faster than waiting for the RTO which has a minimum time of 1 s, thus orders larger than typical RTTs in the Internet.

Flow Control

As highlighted before, packet loss can be due to network congestion, but loss may also happen at the receiver. TCP uses a buffer to store the reconstructed byte stream such that the application using TCP can read it. When the data rate exceeds the application's read speed from this buffer, the buffer will eventually overflow, again leading to losses. Such an overflow can quickly happen when a server sending data has more processing power than a client receiving it. To determine an appropriate rate at which TCP should send while not overflowing a receiver, TCP uses a flow control mechanism. In each segment's header, TCP announces the number of bytes it is capable of receiving. When the buffer fills, TCP advertises that less space is available and when the application has read the bytes from the buffer, TCP signals that more space is available. This signal is called a *window*, and the sender must not send more data than fitting in this window. To this end, TCP keeps track of the unacknowledged bytes that it has already sent, i.e., bytes that will fill the window soon, and bytes that it is still allowed to sent (within the window). Therefore, one refers to this mechanism as a *sliding* window that opens up (for new bytes) when new available buffer size is announced and shrinks when data is acknowledged, thus, in the end, sliding over all bytes.

Connection Teardown

When one party decides to terminate the connection, it can signal that it will not send any further bytes. Similar to the connection establishment, another flag is used (FIN) to indicate that no further bytes will be transmitted. TCP again acknowledges the receipt of this special flag by consuming a sequence number just as in the 3-way handshake. After both parties have signaled that no more bytes will follow, the TCP instances regard the connection as terminated and they can free the connection state, e.g., the TCB. In case TCP receives a segment that it cannot associate to a connection, e.g., after it terminated a connection, but unfortunately, the network duplicated and delayed a packet, or the sender thought the final ACK was lost and retransmitted the packet, TCP uses another mechanism to signal the reception of an invalid segment. To this end, TCP sends an empty segment setting the RST-flag in its header. An RST-receiving TCP knows that the original recipient was unable to associate the packet and can thus also drop all state related to this connection. For

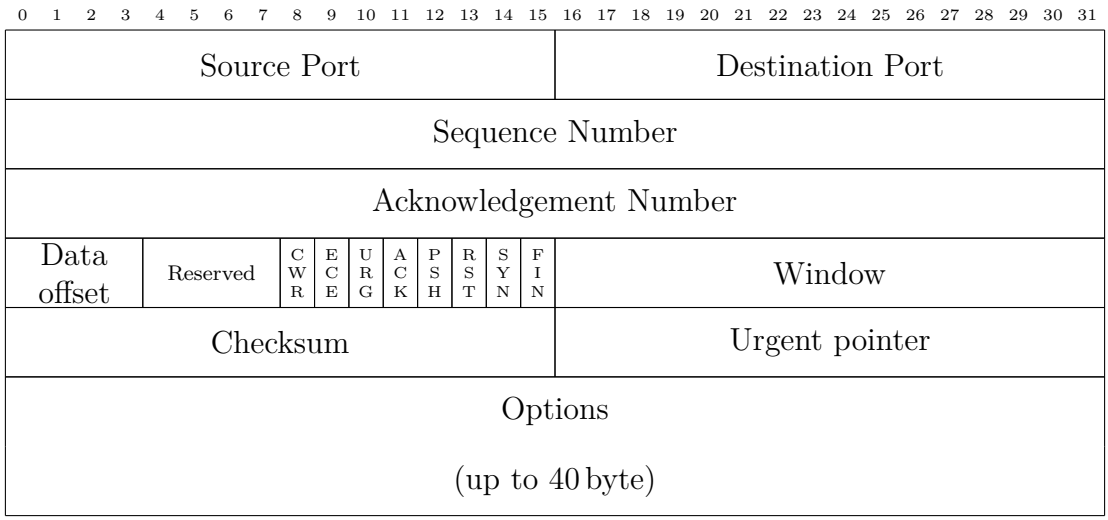


Figure 2.7 The TCP segment header according to RFC 793 and RFC 3168.

this reason, some TCP implementations make it easy for themselves and use RSTs to terminate a connection.

Wire Format

TCP realizes its functionality with the help of a header that we depict in Figure 2.7. Apart from ports identifying application processes on both sides, it contains sequence and acknowledgment numbers. As the TCP specification restricts their space to 32 bit each, TCP must handle overflows correctly. Further, a data-offset identifies where the segment's payload (i.e., the chunked byte stream) starts as the header can contain up to 40 B of options. These options are, e.g., used to signal each endpoint's MSS. A set of bits represents the different flags. For example, SYN for initial segments during the handshake, ACK when an acknowledgment number is present, or FIN for connection teardown. The TCP specification has also left additional space for future extensions to these flags and requires implementations to set these flags to zero when not used and to ignore them when received and unknown. The flow control window is only 16 bit, thus allowing at most 64 kB to be in flight. Today, this is significantly too small to fully utilize high bandwidth links; a TCP option called window scaling can be used to define a multiplicator to this window allowing larger values. TCP further possesses a checksum to check the data and header for correctness as well as an urgent pointer which is rarely used and which we do not discuss further.

X over TCP and the Problems it Creates

TCP is the basis for nearly all Internet communication today with few exceptions. Ranging from simple file transfer applications such as FTP, over BGP sessions to the Web using HTTP on top. TCP is however especially challenged in areas where high interactivity is needed and specifically where losses are actually tolerable. To this end, e.g., online games demanding low latency are typically built on top of

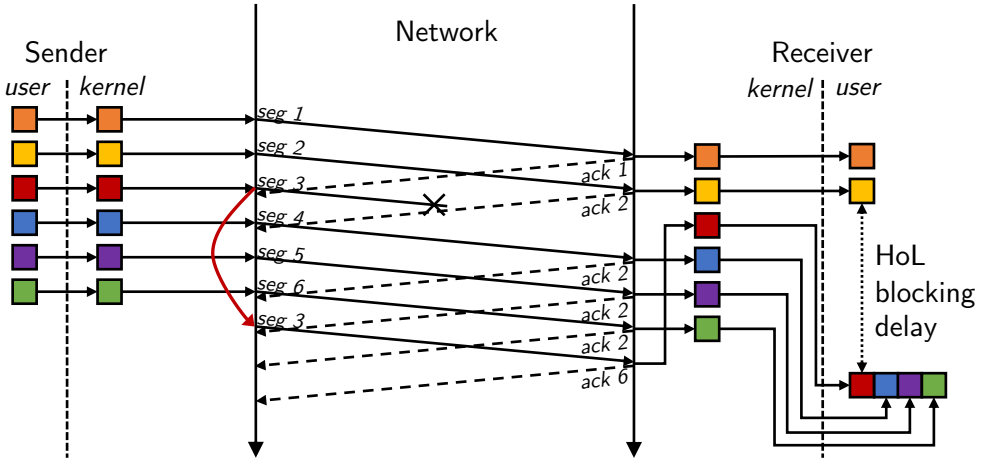


Figure 2.8 HoL in single stream transport protocols: A single lost segment, even though not related to the other data items (color) blocks them on loss until a retransmission arrives and data can be delivered in-order to the application. Please note that the segment numbers and acknowledgment numbers do not follow TCP semantics and should just illustrate the sending and reception of segments. Adapted from [MPF16, Fig. 1].

UDP as they can tolerate loss and do not require high data rates. However, as TCP has been the only practically useable reliable transport protocol that ships with every major operating system, the applications building on top of it have continually evolved and today their mode of operation does not map frictionless to TCP. The most prominent example is HTTP. When the Web evolved, it became apparent that parallelization was very helpful, i.e., being able to request multiple resources in parallel. With HTTP/1.1, browsers thus started opening multiple connections, which is inefficient due to each connection having to perform a handshake and further, as we will later see, have to compete against each other. HTTP/2 solves this inefficiency by allowing browsers to multiplex several requests over a single connection, thus solving the problem of multiple connections but creating a new one. As now independent objects are mapped to a single TCP stream they suddenly become dependent on each other as TCP has no means of differentiating them. Thus now, packet loss prevents TCP from delivering subsequent segments to the application even though they could contain independent resources. This is rooted in TCP trying to bring the byte stream in order and to correct gaps which we illustrate in Figure 2.8. This behavior is generally known as head-of-line (HoL)-blocking. In the figure, the application in user space, e.g., HTTP/2, multiplexes individual objects (represented by different colors) into a single TCP stream. While in practice segmentation of the stream is up to the kernel, and thus objects of different color could mix into a single segment further aggravating the problem, the problem already manifests when assuming that segments only carry individual objects. The loss of Segment 3 blocks Segments 4 to 6 from delivery to user space because TCP's mode of operation does not assume independence of segments. This linkage is not a shortcoming of TCP but rather manifests in the way TCP is used today, i.e., as a transport for individual messages or multiple streams, which were never TCP's design goals. HoL is one of the reasons for many new transport protocol proposals such as QUIC which is the subject of the next section.

2.3.2 QUIC

QUIC, in comparison to TCP, is a new transport protocol for the Internet. It was initially developed at Google and first shipped in Chrome's developer preview in 2013 [Iye15]. Back then QUIC was an abbreviation for Quick UDP Internet Connections, and, as the name suggests, builds on top of UDP. Thus, even though being a transport, it sits above the "traditional" transport layer. Therefore, one usually implements it in user space, which allows updating the transport independent from the operating system promising faster dissemination of new versions.

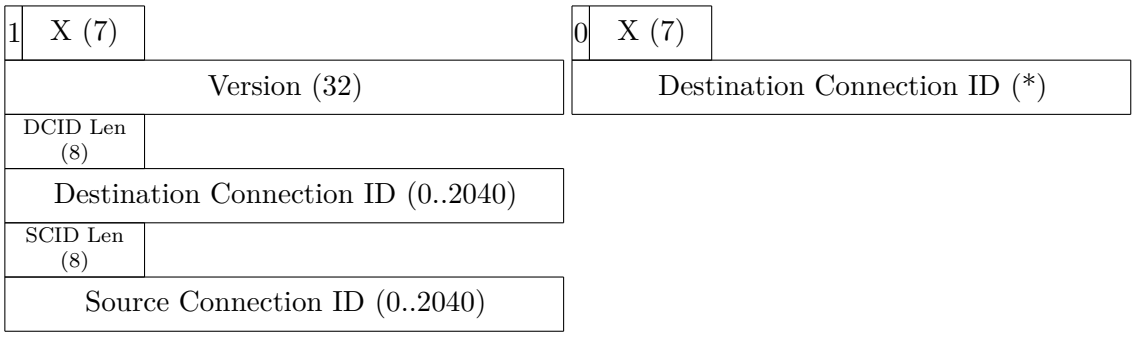
One of the driving reasons for its development was the ossification that network protocols have seen (see Section 2.1). Google themselves have contributed significantly to many transport optimizations, e.g., TCP Fast Open [RCC+11] that allows 0-RTT data exchange in TCP. Nevertheless, the lack of deployability renders many new protocols and extensions practically useless. Thus, QUIC's design goals were to have a deployable and *evolvable* transport that offers TCP-like functionality combining decades of transport protocol research. QUIC tackles these goals by fully encrypting its transport, i.e., including its headers. Thus, only the end-points are able to decrypt the transport layer information, excluding any observer on the path with the goal that no system can eventually ossify around the visible wire image. Within the decrypted headers, QUIC is versioned (in contrast to version-less TCP). It further has the ability to negotiate these versions for backward compatibility.

Similar to TCP, QUIC offers a reliable transport but, in contrast, allows multiplexing individual byte streams over a single connection without introducing dependencies and thus circumvents HoL-blocking. As QUIC is fully encrypted, it does not require TLS on top and is generally able to exchange data after one RTT and can even be used in 0-RTT mode when it has contacted a server before. Moreover, it enables mobility as it does not bind a connection to the IP and port numbers but introduces a separate connection ID, which further allows it to work seamlessly in the presence of NATs. When, e.g., the client's interface addresses change (e.g., when switching from WiFi to mobile data), a server can associate the connection with the help of the ID and the cryptographic context they already established.

After its introduction in 2013 and further development, Google approached the Internet Engineering Task Force (IETF) to standardize QUIC in 2015. As of August 2020, the first IETF version of QUIC is on its last laps towards standardization[2]. When referring to QUIC, we refer to both variants, either Google QUIC (gQUIC) or IETF QUIC (iQUIC). iQUIC offers the same primitives as gQUIC but, while sharing many principle ideas, they are two different protocols that are not compatible with each other. During standardization, gQUIC, as implemented in Chrome, slowly transitions to iQUIC.

Conceptually, QUIC is still very similar to TCP in that it performs retransmissions, has flow control, and uses acknowledgments. Thus, we first take a look at how TCP concepts have been applied in QUIC and subsequently illustrate the connection establishment and version negotiation as a fundamentally different design. Finally, we discuss which problems QUIC faces that were no issues for TCP.

[2]Which was also said a year earlier, so only time will tell the truth in this statement.



(a) The long header (b) The short header

Figure 2.9 The iQUIC invariant headers according to invariants draft-06 [Tho19]. Bits marked with an X are version specific. Numbers in parenthesis denote the possible size of each field in bits. The asterisk signifies that the length is not specified as it has been established elsewhere.

TCP Concepts in QUIC and How They Differ

Similar to TCP, QUIC [IT19] has a header (actually it even has two headers). However, they do not contain a whole lot of data. As the headers are subject to change in future QUIC versions, a minimal set of fields is defined to be invariant, i.e., will never change. We depict the long as well as the short invariant header of iQUIC in Figure 2.9, and we will also continue to illustrate QUIC with the help of the more recent iQUIC. QUIC uses the long header during connection establishment, and once it has negotiated essential parameters such as the connection ID lengths, it switches to the short header. Different QUIC versions can specify other headers that, in the current drafts, are variants of the short or long header; e.g., there is a version negotiation header or an initial header which are both derived from the long header. As visible from the figures, QUIC does not carry ACKs or sequence numbers in its header. Notwithstanding, QUIC does not even possess sequence numbers in the way TCP does. This absence has multiple reasons. First, QUIC carries multiple streams per connection, and thus, a single sequence number could not be used to enumerate different byte streams. Second, TCP uses the sequence numbers, that number the bytes, also for acknowledgments. This mechanism is different in QUIC; in QUIC Version 1, each packet carries a unique and monotonically increasing packet number that is independent of the byte streams. QUIC carries information regarding the order of bytes and ACKs in so-called *frames*.

Frames. The content of a QUIC packet is made up of *frames*, as shown in Figure 2.10. Frames can carry data (i.e., stream frames) or meta-data (e.g., ACK, crypto, or ping frames). Stream frames carry the bytes of the individual streams and can be regarded like the byte stream of a TCP connection. To this end, a stream frame contains a stream ID, an offset where the bytes belong within the byte stream (optional if at the stream start), and if the stream frame does not span up to the end of the QUIC packet, it contains a length field denoting the size of the byte chunk that follows. This information allows QUIC to bring the bytes of individual streams in order before delivering them to the application.

Acknowledgements. Importantly, these offsets are not used to detect packet loss. To this end, QUIC uses the packet numbers and ACK frames. In contrast to TCP,

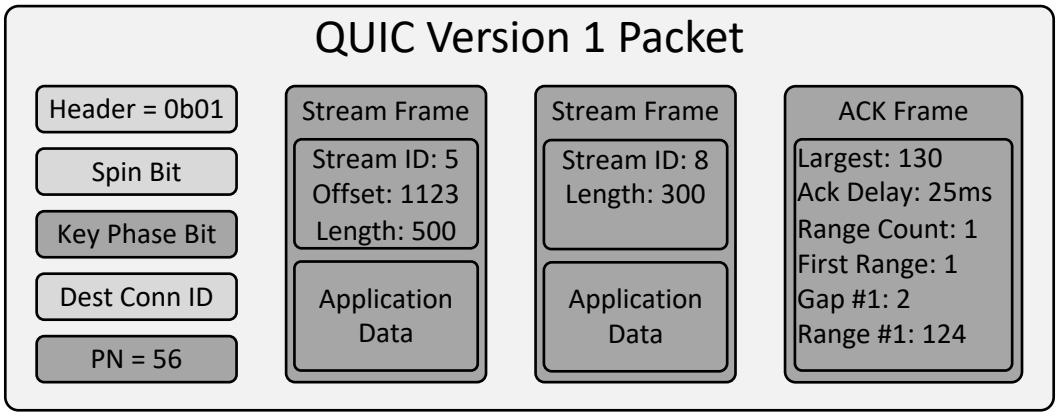


Figure 2.10 Example QUIC Version 1 packet with a short header and two stream frames and an ACK frame. Dark gray parts of the packet are encrypted and thus not visible to a passive observer. Figure adapted from [Iye19].

where retransmissions carry the same sequence numbers (as the original transmission), in QUIC, retransmissions carry a new packet number. This unambiguity allows QUIC to easily differentiate between retransmissions and out-of-order packets[3]. Further, QUIC adopts the concept of SACKs from TCP (but allows up to 256 SACK ranges instead of at most four in TCP) and does not possess a single ACK field anymore. An ACK frame always acknowledges the *largest* received packet number, in contrast to TCP where the last byte before the first gap is acknowledged[4].

Notwithstanding, only announcing the largest received packet number is not helpful alone. A QUIC endpoint uses the *first ACK range* field (see Figure 2.10) to signal how many contiguous packets it received preceding the largest received packet. In the example, this means that QUIC received packets 130 and 129 as the range is only 1. More ranges can follow as denoted by the *ACK range count* field (in the example just one range follows). This range now denotes that there is a gap of three packets, i.e., 128, 127, and 126 are missing, encoded as the value two (because, $128 - 126 = 2$). It furthermore denotes that there are 125 contiguously received packets preceding this gap, thus packets 1 to 125 were received (again encoded as $125 - 1$=124).

Retransmissions. When QUIC detects a packet as lost, e.g., Packet 126 from the example, the stream frames it contained were lost, and, as QUIC is a reliable transport, QUIC must retransmit them. However, as we have already noted, QUIC will not just retransmit the packet with the same packet number, but instead, it uses a new packet number for this. It will not even retransmit all of the packet's contents. For example, it is of no use to retransmit outdated ACK frames, or data of a stream that has been closed in the meantime. To this end, QUIC will only retransmit the data that is still needed by the other end. In this example, QUIC should retransmit

[3]In TCP, a retransmission may look alike the original transmission which makes it difficult to regard a packet as lost and not just out-of-order for appropriate CC actions.

[4]ACK frames further carry a timestamp that denotes how much time has passed since receiving this largest packet number. Similar to TCP, QUIC allows delaying ACKs. The timestamp then allows knowing for how long, enabling more precise RTT estimations as the delay can be subtracted out.

the stream frames in a new packet with a new unique packet number. It could further add a *new* ACK frame with more up to date information if space permits. An ACK frame will eventually notify the sender when the retransmitted content is successfully received. Due to the fact that QUIC never retransmits old packets and only retransmits content, gaps in the packet number space will never close. As the ranges are limited, QUIC will eventually stop reporting specific gaps when it is sure that the other end has successfully received the information about the gap. The QUIC standard does not define how an implementation must do this. Notwithstanding, since every packet is acknowledged, QUIC even acknowledges the receipt of the ACK frame[5]. Thus, each end-point knows when which information was received and can act accordingly and stop reporting this information.

Flow Control. QUIC's flow control mechanism does, in principle, not differ from TCP, i.e., the receiver informs the sender about its buffer status. There is a connection-wide flow control that offers a global limit of bytes it can receive. But further, each stream has an additional unique flow control limiting the stream's bytes. In contrast to TCP, QUIC signals these windows (called maximum allowed data in QUIC) for each flow control with the help of individual frames, and thus, they are not always present. For calculating the still allowed number of bytes that QUIC can send/receive on a stream, QUIC uses the largest sent/received offset in a stream, as potential gaps in front are already committed in the buffer for out-of-order receptions.

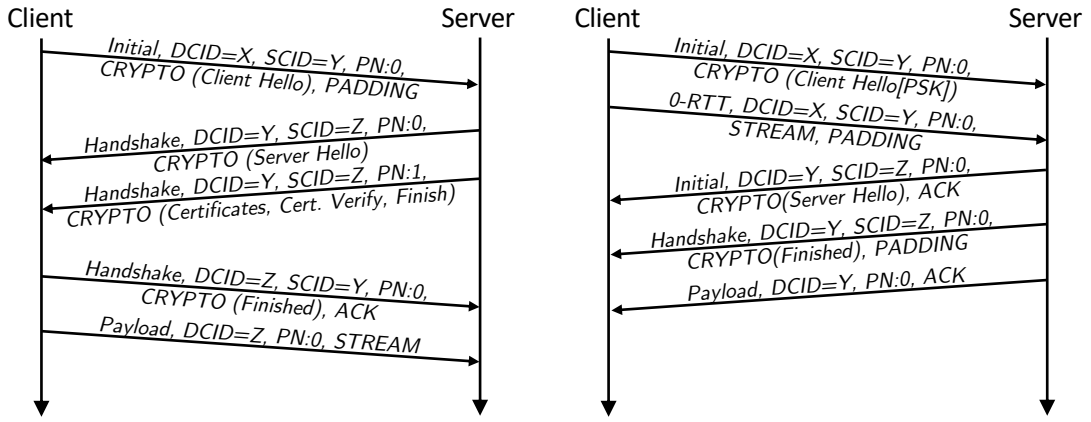
While there are many more details and new concepts in QUIC that are not present in TCP, we refrain from an in-depth discussion here and direct the reader to the RFCs as QUIC literature is scarce. For this dissertation, understanding the motivation and core concepts is imperative. However, as we are going to investigate the use of QUIC in the Internet (Section 3.2), we will make use of the connection establishment and the version negotiation, which we will discuss next.

2.3.2.1 Version Negotiation and Connection Establishment

As QUIC is supposed to be evolvable, it needs a way to negotiate which version of QUIC it wants to use during connection establishment. QUIC does this with the help of the invariant header, which we depict in Figure 2.9a, and the version field contained in it. Thus, when a QUIC receiver receives an attempt to establish a connection, it can inspect the version field to determine if it supports this version. In case it cannot support it, the receiver will send a version negotiation packet[6], which is a long header packet with a zeroed version field having attached a list of 4 B versions that the end-point supports. This list may also contain reserved versions that no QUIC endpoint will ever support in an attempt to get implementors to parse this list correctly.

[5] QUIC has a mechanism to stop it from acknowledging only ACKs such that an ACK ping-pong is prevented.

[6] The version negotiation packet is of course also invariant across all QUIC versions.



(a) QUIC 1-RTT handshake and data exchange after the first roundtrip.

(b) QUIC 0-RTT connection establishment that allows exchanging data within the first roundtrip.

Figure 2.11 QUIC connection establishment using either 1-RTT or 0-RTT setup. Please note that the QUIC packets may contain additional information, especially, transport parameters are exchanged, which we do not display to keep readability.

Informed about the supported versions, a client can select a version it also supports and commence a new connection establishment. QUIC offers two different modes of connection establishment that either take 1-RTT or 0-RTT, as shown in Figure 2.11.

1-RTT Setup. When QUIC has never had contact with an end-point before (e.g., when initially visiting a website), QUIC uses a 1-RTT connection establishment (Figure 2.11a). An *Initial* message carries a TLS 1.3 *client hello* and announces a source connection ID. The server typically replies with two handshake messages, reporting its own source connection ID, and completing the TLS handshake with a server hello and the certificates that are usually larger than a single packet. After this, the client acknowledges the previous message and also send a handshake message[7]. As both parties have now established cryptographic credentials, the client can directly send further messages containing data (carried on a stream), notably these message can already contain the short header.

0-RTT Setup. However, 0-RTT connection establishment (Figure 2.11b) is only possible after having had a previous connection. Now a client can again send an Initial message, yet this time, it can include a pre-shared key from the previous connection. This cryptographic material allows the client to directly send a second message (if space permits, this may even be in the same packet) that contains encrypted 0-RTT data on a stream frame. The server must nevertheless finish the handshake by sending a server hello (certificates must not be transmitted again) acknowledging the Initial message from the client and subsequently sending a *Finish* message completing the handshake. At the same time, the server can answer with payload, e.g., at least acknowledging the client's byte stream.

[7]Please note that Initial, Handshake and Payload messages operate in their own packet number space and thus packet numbers seem to reoccur.

Notably, 0-RTT connections are vulnerable to replay attacks [FG17], which challenges their applicability in the Internet at large. These replay attacks are less of a concern when the packet's content cannot modify any state since the network will not deliver the answer to the attacker, nor would she be able to decrypt the answer. In HTTP, such a request is called *idempotent*. It is however not possible for the transport to differentiate idempotent from non-idempotent data and thus 0-RTT should only be used in safe environments or the application using QUIC must implement replay protection on top. Thus, 0-RTT has the potential of seeding up connections but should be used with caution to not suffer from replay attacks.

Still, there are further challenges to QUIC as we will discuss next.

2.3.2.2 Challenges for QUIC

QUIC is mostly challenged by the fact that it must work on top of UDP which has been playing a neglected role in the Internet since only DNS was heavily using it. We will now highlight four major challenges for QUIC that stem from the fact that it hides all signaling information and is deployed on top of UDP.

QUIC and NATs

As QUIC operates over UDP and must work with IPv4 and thus with NATs, it must handle NAT rebindings. For TCP, a NAT usually tracks the connection establishment and releases a mapping on teardown, in addition to implementing large timeouts, as a safeguard, to eventually release the mapping. As there is no connection establishment in UDP, UDP mappings can only be handled by timeouts when idling. However, Hätönen et al. [HNE+10] found that UDP timeouts often do not follow IETF specifications of being longer than 120 s and are thus more aggressive than those for TCP. QUIC tackles these short timeouts through periodic pings to not idle for too long and alternatively connection migration kicks in, this is, however, only possible when the device behind the NAT sends data and not vice versa.

Transmission Efficiency

Similarly, discovering the available path maximum transmission unit (PMTU) is challenging in QUIC. While it is in principle similarly challenging in TCP, network operators are effectively using middleboxes for PMTU discovery with the help of TCP's MSS option. Middleboxes on path will just lower (clamp) the MSS value during connection establishment to a value that allows fitting segments into the maximum transmission unit (MTU) of their network. While this violates the end-to-end principle, this effectively allows efficient PMTU discovery. In QUIC, middleboxes cannot mingle with the transport headers by design and thus are not able to "help" in this case. Therefore, the QUIC standard suggests a minimum MTU that must be supported by the path and suggests to implement some form of PMTU discovery, e.g., [FJT+19].

Network Monitoring

Network operators thus far have used middleboxes to, in their eyes, enhance and monitor the transport. Notably, the monitoring aspect has given rise to many concerns by network operators. With TCP, operators use packet traces to measure RTTs or losses with the help of the unencrypted headers (e.g., inspecting holes in the sequence number space or measuring the time until packets are acknowledged). In contrast, QUIC hides this information from the operators through encryption. Thus, operators cannot rely on using the network traffic itself to determine the key performance indicators (KPIs) of their network and are effectively operating the network blindly. It took over two years of discussion in the IETF until consensus was reached to expose a single bit to the network, the SPIN bit (see Figure 2.10) that allows a passive observer to determine the RTT. A network operator can track the state of this bit, and client and server either mirror or invert this bit on reception to create a rectangular signal whose period then corresponds to the RTT. De Vaere et al. [DBK+18] proposed to use further bits for loss detection. However, these proposals could not reach a consensus which highlights the difficulty and fear when standardizing a protocol that should not ossify.

Rate Limiting

An additional issue with UDP is that it is sometimes rate-limited by operators. UDP does not implement CC, in contrast to TCP and QUIC as we will discuss in the next section, and can potentially easily overload a network. Operators safeguard this by deploying rate-limiters or shapers that limit a UDP flow's data rate. Nonetheless, as QUIC implements CC, these limits hinder its efficiency. It is mostly unknown how prevalent UDP rate-limiting is in the general Internet, but Google reports [LRW+17] that they found instances of rate-limiting during their QUIC deployment.

As CC today is likely one of the most essential components of transport protocols governing the efficiency and fairness of Internet transport, we are going to discuss it next.

2.3.3 Congestion Control

Congestion Control (CC) is a fundamental building block in today's transport protocols that strongly influences the performance of data transmissions. In contrast to flow control, which protects the receiver, CC was built in the 1980s to counteract the congestion collapse of the early Internet [RFC896], i.e., it is used to avoid overwhelming the network. CC introduces a purely local congestion window (cwnd) that limits the number of unacknowledged bytes in flight to not congest the network. Thus, with CC, a transport may send the minimum of the flow control window and the cwnd. A CC algorithm governs the evolution of the cwnd subject to algorithm-specific congestion signals. Since its inception, CC has been an active field of research and has recently seen promising new developments with BBR [CCG+16b] or Vivace [DMZ+18].

Given the plethora of different algorithms that have been developed, only two have RFCs, NewReno [RFC6582] and CUBIC [RFC8312]. CUBIC is TCP's default CC in Linux and is often the baseline to which other algorithms compare themselves. The basis for these CCs lies in [RFC5681], commonly referred to as Reno, which defines four intertwined mechanisms that together build the basis for CC algorithms: slow start, congestion avoidance, fast recovery, and fast retransmit. We will discuss slow start and congestion avoidance as they are fundamental for safe transport operation.

Slow Start

At the start of a connection, the two end-points are generally unaware of the path that their packets will take through the Internet. Thus, they do not know the available bandwidth or latency, and even if they had a previous connection, the path might have changed in the meantime. Transports use slow start, in contrast to what its name suggests, to quickly probe the path for bandwidth. To this end, slow start grows the cwnd exponentially by increasing the cwnd by the number of acknowledged bytes in every roundtrip. This increase effectively doubles the cwnd after each roundtrip. One fundamental property of slow start is the initial cwnd (IW), i.e., the size of the cwnd when the connection starts, as it sets the basis for the exponential growth which we investigate in detail in the next chapter. [RFC5681] defines the IW to be between two to four MSSes (depending on the exact segment size), but other RFCs exist that recommend different values. A transport protocol leaves slow start when it detects congestion or reaches a predefined threshold (which is typically derived during a connection and new connections can be seeded with it). Depending on the exact algorithm, congestion can be measured differently, e.g., CUBIC uses loss as a congestion signal; in contrast, BBR uses increases in delay as a signal. Notwithstanding, when congestion is detected, the transport generally switches to congestion avoidance.

Congestion Avoidance

In congestion avoidance, the algorithm tries to slowly probe for additional bandwidth while trying to minimize its aggressiveness. When it detects congestion, it reduces the cwnd. Since congestion may resolve when other flows leave a congested link, a CC must also constantly probe for newly available bandwidth, typically by increasing the cwnd. To this end, a popular class of congestion avoidance algorithms uses additive-increase multiplicative-decrease (AIMD) in congestion avoidance, which is characterized by a linear increase and exponential decrease on congestion, resulting in the well-known cwnd sawtooth behavior of, e.g., TCP.

In the following, we describe three popular algorithms, i.e., Reno, CUBIC, and BBR.

2.3.3.1 Reno

Reno [Jac90] is an enhancement to Tahoe [Jac88], the first CC algorithm, but they only differ in small details that however have significant performance impacts. Reno

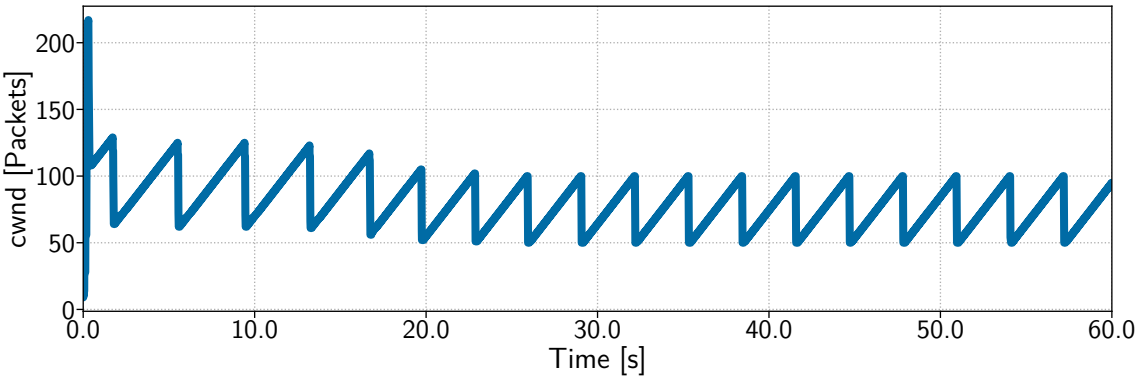


Figure 2.12 Reno's cwnd in a network with a min. RTT of 60 ms, a queue size of 50 packets and 50 Mbit/s link speed.

uses packet loss as a congestion signal in TCP. It further uses timeouts (RTOs) as a signal for *major* congestion and duplicated ACKs as a signal for *minor* congestion.

Slow Start

Reno's slow start basically follows the generic slow start that we explained earlier. During slow start, Reno grows the cwnd exponentially by increasing it in every roundtrip by the number of acknowledged bytes. It exits slow start when reaching a threshold (the slow start threshold (ssthresh)) or when any form of congestion, i.e., major or minor, is detected.

Congestion Avoidance

In congestion avoidance, Reno grows the cwnd by one MSS (additive-increase) in every roundtrip, and when detecting major congestion, it sets ssthresh to half of the bytes in flight but at least to two MSSes. It then switches back to slow start setting the cwnd to an IW of one MSS. Similarly, when the connection has idled, i.e., no bytes have been transmitted for at least one RTO, it also goes back to slow start. However, when minor congestion is detected during congestion avoidance, Reno departs from Tahoe's behavior and switches to so-called *fast retransmit*.

Fast Retransmit and Fast Recovery

A TCP receiver should generate duplicated ACKs when receiving out-of-order segments. Out-of-order segments can mean two things. First, the segments are reordered, or second, a packet was lost, and thus, all following packets show a gap and appear out-of-order. Fast retransmit interprets three duplicated ACKs as a signal for loss which the sender should repair. On the first two duplicated ACKs, the sender may release two new segments when the bytes in flight are still below the cwnd but allows overbooking the cwnd by two additional MSSes. On the third duplicated ACK, it sends the segment that appears lost without waiting for the RTO. Further, it sets ssthresh to half of the bytes in flight as before. Reno then switches to *fast*

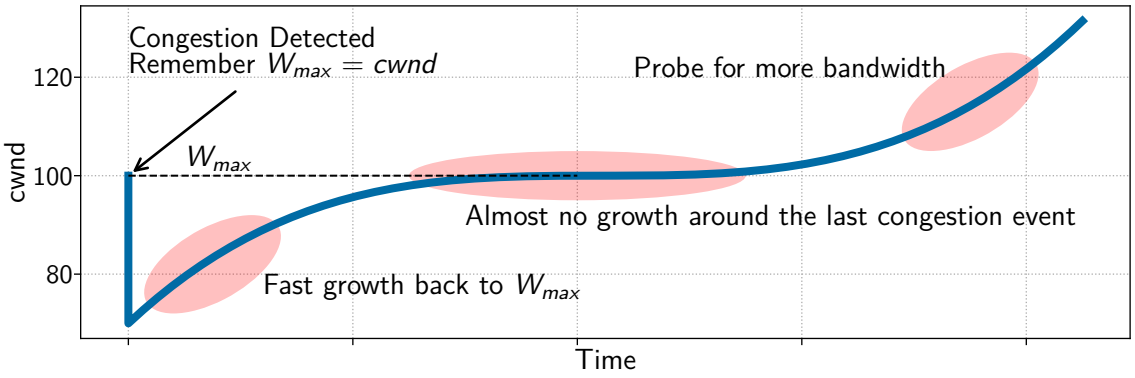


Figure 2.13 Evolution of the cwnd in CUBIC according to its cubic function. After a minor congestion event, the cwnd is reduced by a factor of 0.7. Subsequently, the function quickly probes for bandwidth. Around the last congestion event, there is nearly no increase. Finally, CUBIC again starts looking for bandwidth.

recovery to repair further losses until a non-duplicated ACK is received. TCP does this by inflating the cwnd to ssthresh plus three segments (i.e., the segments that have generated the duplicated ACKs before) to release further segments. For each additional duplicated ACK received after the three, the cwnd grows by one MSS allowing to potentially release new data. Finally, fast recovery ends when previously unacknowledged data is acknowledged setting the cwnd back to ssthresh (effectively halving the previous maximum bytes in flight before the loss, i.e., multiplicative-decrease) to deflate the overbooked cwnd again and then switches back to congestion avoidance. This results in Reno's characteristic sawtooth as visualized in Figure 2.12.

Reno defines a basic set of algorithms that allow safe transport operation. However, it may suffer from a couple of inefficiencies. To this end, NewReno is a modern enhancement to Reno that better handles the presence of SACKs which are very common in today's TCP implementations. We continue to look how CUBIC modifies Reno in light of high bandwidth networks.

2.3.3.2 CUBIC

Reno's linear increase by one segment in congestion avoidance is often too small for networks that have a high bandwidth delay product (BDP), i.e., either a high bandwidth, high delay, or both. For example, when bandwidth becomes available (e.g., due to another flow leaving), Reno's growth is too low to utilize the network efficiently. CUBIC [RFC8312] in that regard modifies several parts of Reno; most important is the change in congestion avoidance that lends CUBIC its name. It uses a cubic function (visualized in Figure 2.13), i.e., having a steep (concave) incline followed by a plateau where the function changes the profile to a convex incline to further grow the cwnd. It still performs multiplicative-decrease but remembers the previous cwnd to calculate a function that has a plateau at exactly the previous cwnd (W_{max} in Figure 2.13). Thus, CUBIC grows super linear towards the previous cwnd and probes for additional bandwidth using its convex profile.

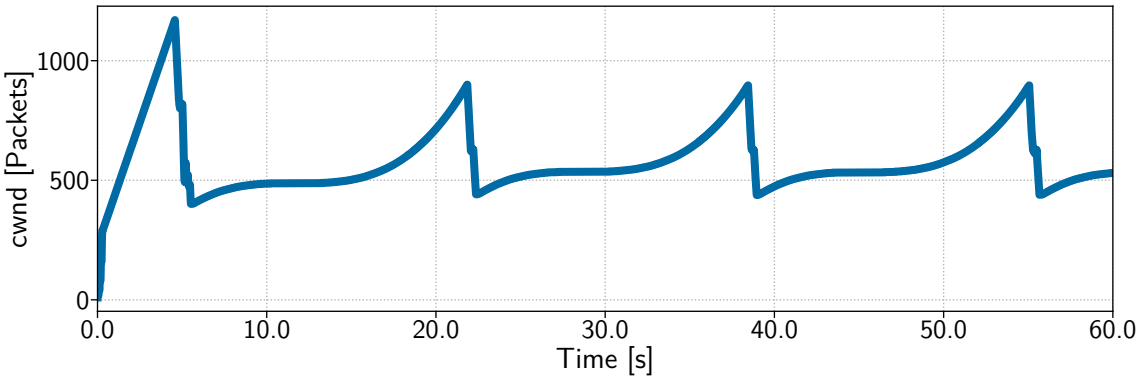


Figure 2.14 CUBIC's cwnd in a network with a min. RTT of 60 ms, a 200 packet queue and a 50 Mbit/s link.

Further, CUBIC modifies the multiplicative-decrease factor by which the cwnd is lowered from 0.5 to 0.7, which makes it more aggressive. We visualize CUBIC's cwnd evolution in Figure 2.14. CUBIC is designed in a way to perform similarly to Reno in networks with smaller delays, and thus when CUBIC's increase is less than that of Reno (which can happen with small RTTs), CUBIC falls back to Reno's behavior. There are additional suggested changes in CUBIC that further alter its behavior in comparison to Reno, e.g., it may use a limited slow start [RFC3742] or hybrid slow start [HR08] which are both designed to be less aggressive than the traditional slow start.

CUBIC, as well as Reno, use loss as a congestion signal. We will now look at BBR that uses delay as a congestion signal.

2.3.3.3 Bottleneck Bandwidth and Round-Trip Propagation Time

Loss-based CCs have an inherent weakness, i.e., they operate at the upper end of the network bottleneck's buffer. As filling the buffer increases the delay, loss-based CCs, by design, maximize the delay in the bottleneck buffer. During AIMD they continually fill the buffer until it overflows and a packet is lost; after the multiplicative-decrease, the process starts again.

In contrast, delay-based CCs try to operate at the lower end of the buffer and use an increase in delay (i.e., filling of the queue) as a signal for congestion. While filling the buffer allows one to increase the cwnd further, data is not transmitted faster as a router must stores it in its queue. In an ideal world, a buffer would only contain a single packet at every point in time. This allows high utilization as there is always a packet to send, but at the same time, low latency as there is at most one packet transmission worth of waiting time.

Bottleneck bandwidth and round-trip propagation time (BBR) [CCG+16b][8] in this regard tries to model the network's path and the bottleneck by estimating the true minimal RTT and the bottleneck bandwidth, i.e., the maximum data rate offered by

[8]We regard BBRv1. BBRv2, as released in July 2019, is out of the scope of this dissertation.

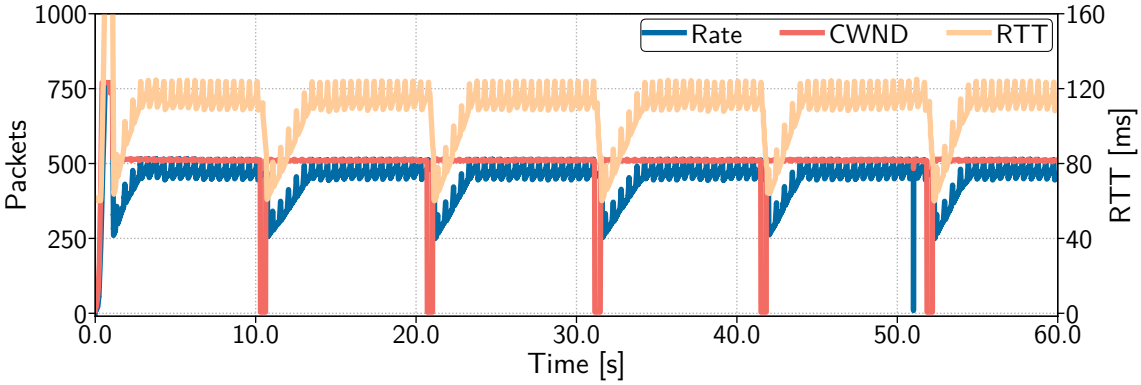


Figure 2.15 BBR's cwnd, packet rate and RTT in a network with a min. RTT of 60 ms, a 1000 packet queue and a 50 Mbit/s link. BBR's bandwidth estimate is derived from the packet rate and the RTT.

the network's bottleneck. Assuming both are known, the optimal cwnd is the product of both, and then one can release the cwnd with the rate of the bottleneck. This data rate, in theory, results in precisely the ideal behavior. Figure 2.15 shows the typical behavior of a BBR flow; the cwnd is rather constant, and BBR constantly probes the bandwidth. Please note that BBR is, by default, configured to overestimate the BDP to better cooperate with loss-based CCs. In the following, we discuss how BBR deals with challenges in real networks such as variable delays, the co-existence with other (loss-based) CC algorithms, and changing network paths.

Estimating the max. Bandwidth: ProbeBW

To calculate its sending rate, BBR counts the bytes that it is able to transmit in an RTT: $rate = \frac{\text{bytes acknowledged}}{RTT}$. At a certain point, regardless of the cwnd, the rate will not increase any further as BBR has met the bottleneck's bandwidth. BBR uses this observation during congestion avoidance in what is called *ProbeBW* to increase the sending rate for one roundtrip every eight roundtrips (see the continuous up and down of the rate in Figure 2.15). If this increased sending rate results in quicker delivery of all bytes (i.e., the bottleneck did deliver it at that rate), new bandwidth has become available and then BBR assumes this to be the new bottleneck bandwidth. In case the sending rate did not increase, BBR has built up a queue in the bottleneck (it did send too fast) and in turn reduces its rate for one RTT to drain the queue it created, before returning to the original rate prior to the probing.

Estimating the min. RTT: ProbeRTT

Further, BBR needs to reevaluate the RTT from time to time. To this end, it uses a method called *ProbeRTT* when it has not detected a reduction in the RTT for 10 s (see the drops in RTT to min. RTT in Figure 2.15). To investigate if a queue has formed, it reduces the cwnd to four packets for at least 200 ms (see the drops in cwnd in Figure 2.15). Since it does not change the rate at which packets leave the sender, it will just drain the bottleneck queue, which allows minimum RTT estimates. When BBR thus detects that the queue was full, it transitions back to ProbeBW as

it must likely reduce the rate. Otherwise, the queue was empty, and there is likely more bandwidth available. When this happens, it switches to *Startup*.

Slow Start: Startup

Startup replaces slow start but mimics its behavior. BBR adjusts the data rate and the cwnd by a factor of $2/ln(2)$ which allows smoothly increasing the data rate by a factor of two every RTT, thus closely matching the original slow start growth. During this increase, BBR searches for a plateau in the delivery rate, i.e., the point where increasing the rate and the cwnd does not yield improvements in the delivery rate. At this point, BBR has reached the bottleneck rate; algorithmically BBR exits Startup when there is no increase above 25 % within three roundtrips (the plateau). However, Startup may have overshot the actual bottleneck bandwidth, and hence, it switches to the *Drain* phase to remove any queue that it might have built.

Drain is similar to ProbeBW as it changes the data rate. However, it now uses a data rate that is $1/(2/ln(2))$, i.e., the inverse of Startup, to remove a standing queue. As BBR already has an estimate of the bottleneck bandwidth and the min. RTT (from the 3-way handshake or a smoothed estimate generated during Startup), it can also calculate an estimate for the BDP. BBR leaves Drain when the delivery rate has reached the bottleneck bandwidth and when the bytes in flight are smaller or equal to the BDP and switches to ProbeBW.

The BBR draft [CCY+17] goes into great depth on how estimated values should be filtered and on how the cwnd should overestimate the BDP to work well on realistic Internet paths. One central property of BBR is that it controls the data rate rather than only controlling the bytes in flight, controlling a sender's data rate is generally known as *pacing* in CC, which we investigate next.

2.3.3.4 Burstiness and Pacing

Most CC algorithms immediately release new data upon ACK arrival. This mechanism builds on top of the observation that a transport eventually generates an ACK-clock. When a large number of packets is sent, they are eventually shaped by the bottleneck's departure rate. Thus, the receiver will generate ACKs corresponding to this rate and eventually this also smoothes the sending of new segments.

Still, when large numbers of ACKs arrive shortly after each other, a CC releases a *burst* of data. This can happen in the first round of slow start, when no ACK-clock is available since the protocol did not exchange any data yet. It may also happen when a sender reinitiates sending after a period of idle (e.g., when using DASH video streaming) or when multiple ACKs are compressed to a single ACK (a "feature" of some middleboxes). These busts are undesired since they may cause packet loss in the intermediate buffers and have led to increasing buffer sizes (which may cause higher latencies).

To counteract this burstiness, a CC or transport protocol can thus use *pacing* to spread out the departure of data over a more extended period. As we have seen

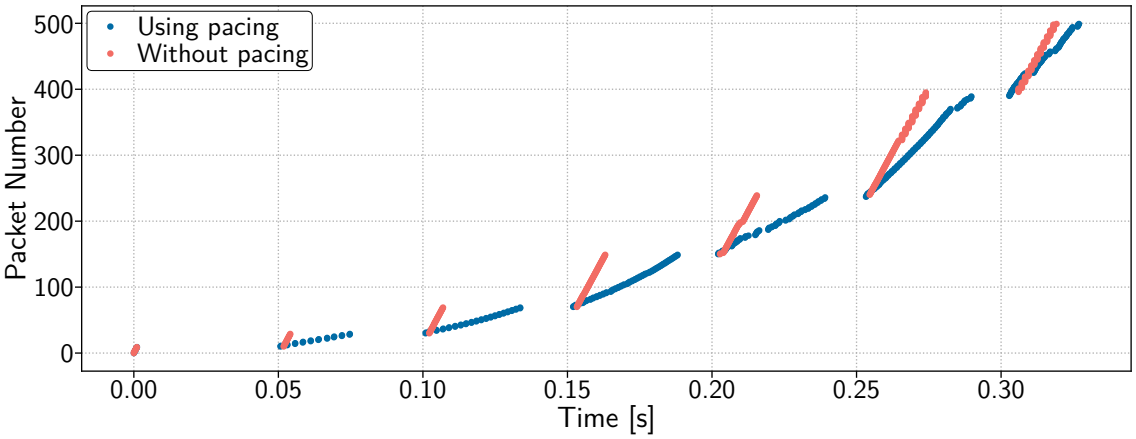


Figure 2.16 A paced and an unpaced Linux TCP flow at 50 ms RTT. Each dot represents a packet sent. The paced flow is not as bursty as the unpaced flow. Please note that both flows burst the first ten packets and that the pacer is configured to only distribute the packets over a part of the RTT, i.e., the Linux default behavior.

before with BBR, BBR releases packets such that it meets the bottleneck's data rate and thus, in essence, it actually paces the data.

Figure 2.16 visualizes a paced and an unpaced TCP flow using CUBIC shortly after connection establishment. Without pacing, a transport releases its cwnd only bound by the central processing unit's (CPU's) or network interface card's (NIC's) processing speed. Assuming a typical 10 Gbit/s server NIC, data is released such that packets have an inter-packet spacing corresponding to the 10 Gbit/s data rate. As visible in the figure, the unpaced flow shows bursts of packets at the start of each roundtrip. In contrast, the paced flow spreads the packets throughout the RTT. Only after several RTTs, the ACK-clock starts evening out the bursty nature of the unpaced flow.

When unpaced data hits a bottleneck, e.g., at an end-user that only connects at 50 Mbit/s, the buffer in front of the 50 Mbit/s link must be able to absorb the burst, i.e., it must be bigger than actually necessary. Generally, it can be advantageous to have a slight burstiness in the data depending on the used medium access and technology. For example, wireless links such as 802.11 perform frame aggregation, i.e., they take, for example, two packets and transmit them as one frame over the channel, increasing the efficiency. When now packets are not available for aggregation due to too large inter-packet gaps, the efficiency might go down. Thus, depending on the access technology, it makes sense to use pacing but to pace trains (e.g., pairs) of packets.

In Section 2.3.4, we take a look at queueing and buffers in general but before, we regard CC fairness.

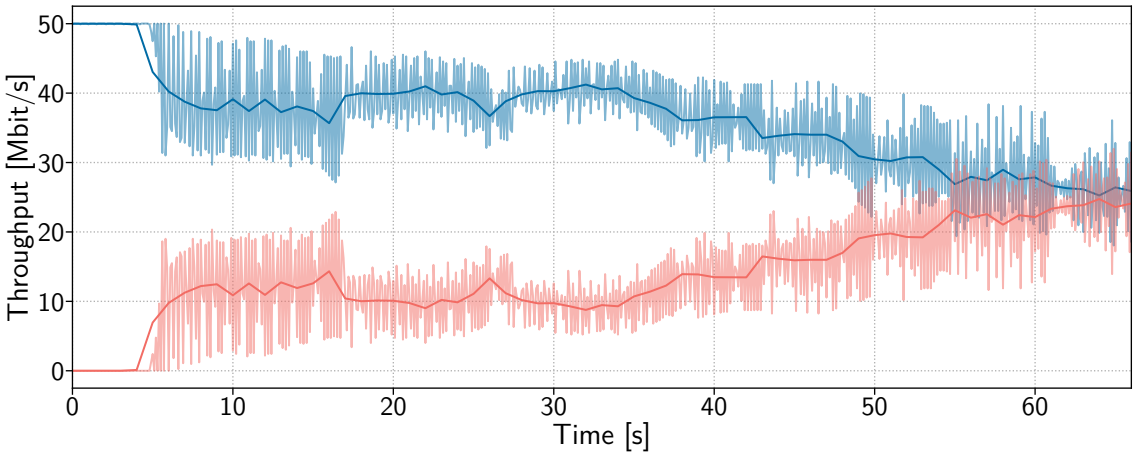


Figure 2.17 Two CUBIC flows compete for bandwidth, a red flow enters after the blue flow has already occupied the 50 Mbit/s link. The darker, more focused lines, show a 1 s rolling average of each flow's rate, while the lighter colors denote a rolling average over 100 ms. It takes over 60 s for both flows to roughly reach an equal share of the available bandwidth.

2.3.3.5 Fairness

CC governs the utilization of a link and its buffers. When multiple flows share the same bottleneck, it is generally desirable to have them share the resources in a fair manner.

This resource allocation usually boils down to a *flow-rate* fairness, i.e., all flows at that bottleneck should have a fair share of the available rate. Flow-rate fairness is not a good metric for fairness in general as it does not recognize the nature and purpose of the flow [Bri07]. For example, the flow-rate of a short 1 s flow compared to a long-running flow of several hours should most likely get more than half of the bandwidth for this short time. This is motivated by taking a look at the utility of both flows. The short flow is likely interactive and requires swift processing, i.e., when it takes 2 s to complete, it may already offer bad performance. While it is likely not relevant for the long flow that takes several hours if it is inflated by an additional second. Thus, Briscoe [Bri07] argues that one should judge the fairness by the *costs* that are induced by the individual flows, e.g., costs could be loss or delay induced to the other flow. Using costs turns out to be practically challenging as there is generally no means to quickly establish them, which is the reason why most research still judges fairness according to flow-rates as it is a simple and intuitive, yet often misleading, metric.

In this dissertation, we also follow the notion of flow-rate fairness, as we have no better means of judging the costs involved. Typically, CCs are judged with regard to their TCP friendliness, i.e., how they behave when they compete with CUBIC (as Linux's default) or Reno (as the IETF's baseline). Figure 2.17 shows the flow-rates of two CUBIC flows. One has already hogged all bandwidth (blue) when a second flow enters (red). We observe that, over time, both rates equalize; nevertheless, it takes over 60 s to do so. In this example, both flows follow the same path, and thus, their transmission latency is the same. Generally, when two flows experience the same

RTT, they should get exactly the same share of the available bandwidth. When the RTTs are different, many CCs share their rates linearly proportional to the inverse of the RTT ratios. This observation is rooted in the fact that each CC can only change the cwnd roughly every RTT since it simply takes so much time until it is aware of any change, e.g., detect losses. Thus flows with a shorter RTT can react quicker to loss (i.e., reducing their cwnd) but also get swifter feedback (i.e., ACKs increasing the cwnd). Whether or not such a bandwidth sharing is desirable is debatable. This relation means that flows with a large RTT typically get a smaller share than flows with a short RTT.

Interestingly, Hock et al. [HBZ17] have shown that BBR does not follow this principle and flows with a large RTT typically (at least when the buffer is large enough) get a higher share, because the calculated BDP is also higher. This also show that fairness critically depends on the buffer size and queuing discipline; we will revisit both in the next section.

2.3.4 Router Queues

Each router requires a buffer, i.e., a fixed size of memory. First, to store incoming packets, and subsequently, after deciding on the next-hop, to enqueue the packet to an egress buffer where each packet must wait for transmission. Buffers allow absorbing temporary mismatches in data rates and small bursts. Their size limits the burst extent and effectively influences the link's utilization and the queuing delay.

Looking at a small example, assuming a 10 Gbit/s link and a buffer of size 10 MB, it takes 8 ms to send all bytes it contains. Thus, one could argue that the maximum queuing delay is rather short. However, using a 10 MB buffer for a 1 Gbit/s would lead to a maximum queuing delay of 80 ms. Thus, this would inflate the maximum queuing delay by a factor of ten. Within these 80 ms, light in a fiber optic cable could travel 16 205 km, thus roughly from New York to Sydney. Yet, if the packet is stuck in the buffer, it travels 0 km in the same time which is obviously undesirable. It is thus critically important to chose appropriate buffer sizes.

2.3.4.1 Buffer Sizing

There have been many works on buffer sizing. We will shortly go through the three most important findings following the summary presented in [GM06].

Rule of Thumb: BDP

The rule of thumb states that a buffer size should be sized according to the BDP,

$$B = RTT \times C$$

i.e., the RTT (two way propagation time) times the link's capacity (C). This relation intuitively stems from the observation of a single TCP flow as we have shown in

Figure 2.12. The cwnd follows the sawtooth, and a TCP can send exactly B bytes in this period, i.e., the distance from peak to peak of the sawtooth. Thus it requires exactly a buffer of size B to never run out of packets, i.e., have 100 % link utilization. Villamizar and Song [VS94] have shown that for a small number of flows with the same RTT (up to eight in their experiments), the loss events synchronize for all flows (i.e., their sawteeth synchronize as well). This means that they will lower their cwnd at the same time, leading to a temporarily underutilization of the link's ingress queue. Given that the globally observed sending pattern (i.e., their sawteeth) is precisely similar, the same (large) egress buffer is still required as in the single flow setting to keep the link utilized.

Small Buffer Rule: BDP/$\sqrt{N}$

The small buffer rule by Appenzeller et al. [AKM04] challenges this observation for a larger number of long-lived flows. Here, the flows do not synchronize, causing their sawteeth to add up to a constant flow of data. They show that it is sufficient to reduce the rule of thumb by the factor $1/\sqrt{N}$, where N is the number of long-lived flows going through the buffer. Knowing N in practice is challenging and can likely only easily be applied to Internet core routers, where the number of long-lived flows is huge. Taking a look back at our example from earlier, the 1 Gbit/s link would require exactly 10 MB buffer when we also assume an RTT of 80 ms. Additionally, assuming that 100 long-lived flows go through the queue, the buffer size reduces to only 1 MB, and thus, the maximum queuing delay also goes down to 8 ms, in turn, light in fiber would only allow traveling from New York to Minneapolis.

Tiny Buffer Rule: $\mathcal{O}(\log W)$

The tiny buffer rule stems from the dream of all-optical switches, i.e., switches that do not convert optical signals to electrical signals and back. Building buffers for such switches is technically challenging, and tiny buffers are thus desirable. Enachescu et al. [EGG+06] have shown that buffers in the order of only tens of packets are possible when sacrificing some utilization.

However, they demand that the flows are non-bursty, i.e., they perfectly pace their packets across their RTT. In this setting, they show that a buffer size in the order of

$$\mathcal{O}(\log W)$$

with W being the maximum cwnd size of the flows going through the buffer, is possible. Ganjali and McKeown [GM06] report from experiments where a buffer of only 20 to 50 packets for a 1 Gbit/s link under realistic workload was enough and only a small performance degradation was observed. They further argue that (core) networks are typically *not* operated at 100 % utilization, and thus, the performance penalty would be even smaller.

All of these sizing rules assume a simple drop-tail queue. Today, there are more advanced queuing disciplines that themselves can help in reducing delay and congestion.

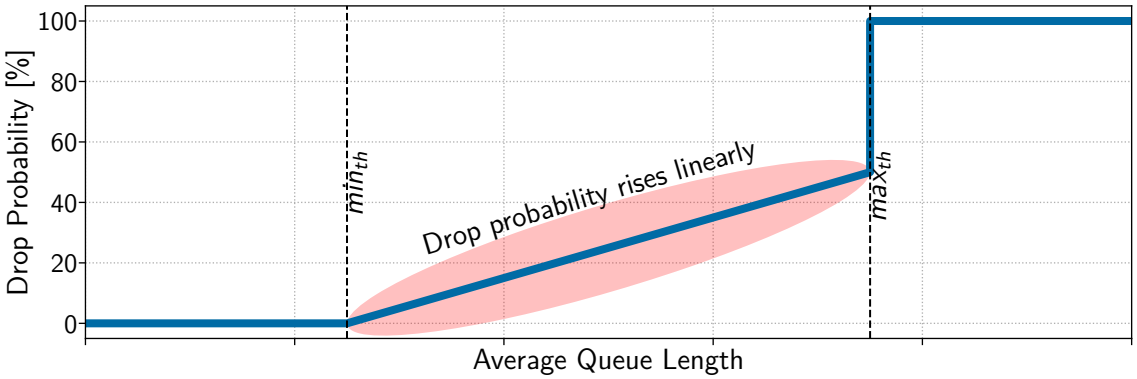


Figure 2.18 RED drops packets starting from a lower threshold (min_{th}). The probability of a drop then increases proportionally with the average queue length; starting from an upper threshold (max_{th}), all packets packets are dropped.

2.3.4.2 Active Queue Management

As we have seen, sizing router buffers is not simple and involves knowing where the buffer is used and what traffic is to be expected. This difficulty is expressed in what is known as bufferbloat [GN12], i.e., the deployment of overly large buffers. Router manufacturers have traditionally sold rather large buffers as link utilization was a key metric that should be optimized. Only the recent years have fostered the understanding that low latency is crucial for many modern Internet applications.

To this end, Active Queue Management (AQM) tackle the problem from another angle and replaces the simple FIFO drop-tail queue with a queue that follows an algorithm in deciding when to drop a packet, sometimes also referred to as a smart queue. AQMs thus allow congestion management.

There have been various proposals for different AQMs. We shortly introduce the ideas behind the first AQM called RED, then look at the state-of-the-art with CoDel and finally regard how we can achieve a *fair* queuing.

RED

Random Early Detection (RED) [FJ93] is an algorithm that uses the average queue size to determine a probability to mark or drop a packet[9]. The idea is to keep the average queue size at a certain level to ensure utilization while keeping an upper bound on the queuing delay. Figure 2.18 shows RED's drop behavior subject to increasing average queue length. Starting from a lower bound, RED increases the drop probability in proportion to the average queue length. When reaching an upper bound, every packet is dropped or marked.

This increasing drop probability seems intuitive; when the queue fills, more congestion happens, and thus, more packets should be dropped to give feedback to the end-hosts. The calculation of the average queue length is, however, rather involved as short time

[9]Marking is used with explicit congestion notification (ECN) to signal congestion instead of immediately dropping the packet.

bursts should not significantly impact the average queue size. To this end, Floyd and Jacobson propose a low-pass filter to determine the average queue length. Further, while the idea is to increase the drop probability, drops should be evenly placed to avoid a global synchronization of the flows as well as a bias towards individual flows. To this end, the actual probability calculation further includes the number of non-dropped packets since the last drop.

Luckily, RED is one of the few AQMs that are available in commercial routers. On the downside, it suffers from a high degree of parameterization (e.g., low-pass weight and thresholds), making it practically difficult to tune RED for a specific workload. CoDel tries to overcome these issues by providing a "knob"-free queuing discipline.

CoDel

Jacobson himself (one of RED's creators) later acknowledges [Jac06] that average queue length is an inadequate measure for congestion. He notes that small differences in the TCP implementations, while all being correct, result in vastly different average queue sizes and concludes that average queue length bears no meaning for demand or load. To this end, Nichols and Jacobson [NJ12] present Controlled Delay (CoDel) (pronounced "coddle") that provides a tuning-free algorithm that they build on the notion of *good* and *bad* queues. This notion basically follows the same rationale as BBR, i.e., sending more than the BDP will only cause a standing queue and thus latency in steady-state. Generally, a queue is *bad* when it is a standing queue, i.e., always having a size above 0 or 1 packet. A *good* queue, in contrast, is a queue that may temporarily be high (e.g., due to a burst in slow start) but then wholly drains to 0 or 1 packet within an RTT (the ACK clock will control the sending rate).

CoDel tries to detect these bad queues and drop packets to reduce the flows' rates such that their cwnd closes in on the BDP. The authors argue that a persistent (i.e., bad) queue can be tracked by observing the packets' sojourn time, i.e., the time the packets have spent in the queue. A key is to track the (local) minimum sojourn time and not an average because if there is a packet with no queuing delay, there cannot be a *persistent* queue (but the average queue length could still be high). Hence, CoDel tracks the minimum sojourn time over an *interval*, and if it does not stay below an acceptable minimal sojourn time (called *target*) within this interval, the queue is considered a bad queue. The interval thus denotes the time it takes to drain the largest burst, which in turn relates to a flow's RTT and thus is set according to the maximum expected RTT across the connections.

Therefore, it is not entirely true that CoDel has no knobs. Jacobson however shows that a target of 5 ms (actually ideally 5 % to 10 % of the RTT) and an interval of 100 ms (actually the maximum RTT) work well over a wide range of link rates, RTTs, and traffic patterns.

With the help of the local minimum sojourn time, CoDel can detect a persistent queue, but it still needs to act to drive it towards *target*. CoDel increases its drop rate slowly until the persistent queue goes away. To this end, when a persistent queue is detected, it calculates the next drop time as

$$next\_drop\_time = t + interval/\sqrt{count}$$

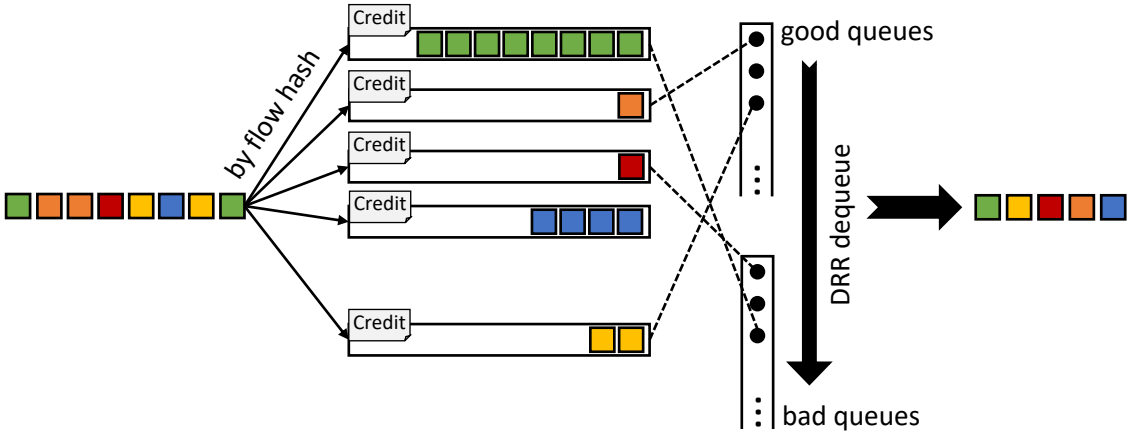


Figure 2.19 FQ-CoDel workflow. First, a flow is hashed to a queue. Queues are either good or bad. Good queues are served first using DRR and dequeuing itself is handled via FQ-CoDel. Good queues can become bad queues and vice versa.

with t being the time of the last drop and *count* denoting the number of drops since dropping started. Thus, when the number of already dropped packets rises, the next drop times get shorter and shorter to give more feedback when it takes more time to get to *target*. CoDel stops dropping packets when the sojourn time meets the target.

More rationals behind CoDel can also be found in [RFC8289]. However, CoDel or RED do not guarantee a fair packet scheduling, which we investigate next.

Flow Queuing using Deficit Round-Robin

So far, all queuing disciplines disregard the bandwidth share of individual flows going through them. For example, RED will drop packets when the average queue length increases for all flows even though some of them might not even contribute to a standing queue.

To this end, we regard *one* way of reaching a fair queuing that is used in flow queuing (FQ)-CoDel [RFC8290], which ultimately uses deficit round-robin (DRR) [SV96]. First, one ideally needs exactly one queue per flow (or per whatever one wants to reach fairness against). This requirement becomes practically challenging, and thus, one usually takes a fixed set of queues and uses hashing to enqueue flows into these queues.

The main idea is now, as in DRR, that each queue has a *byte credit* (with a fixed initialized size) that it is allowed to dequeue whenever it is this queue's turn to transmit packets. When the *byte credit* reaches zero, the algorithm adds a *quantum* to the queue's credit, but it has to relinquish its dequeuing opportunity, and another queue may transmit bytes worth its credit. Thus, the algorithm serves each queue in a round-robin fashion but caps the number of bytes that it dequeues.

At this point, FQ-CoDel (visualized in Figure 2.19) departs from classic DRR and adds another hierarchical layer that classifies the different queues. FQ-CoDel differentiates between *old* queues that have built a backlog and *new* queues that

did not transmit packets in the last round. This differentiation allows prioritizing short flows over long flows. At first, FQ-CoDel takes a look at the head of the new queues list. If the queue's credit is negative, i.e., it recently transmitted all its credit, FQ-CoDel adds the quantum and moves the queue to the tail of the old queues list. Otherwise, it serves the queue, and once it served all new queues, FQ-CoDel continues with the old queues in the same fashion, i.e., if the credit is negative, FQ-CoDel adds it to the end of the old queues again adding the quantum. The exact dequeue routine for each queue is just CoDel itself. If CoDel finds that an old queue has no packets to dequeue, i.e., the queue is empty, it moves it to the end of the new queues, and the process starts over.

Adding the queues to the end of the respective lists ensures that other queues do not starve and that FQ-CoDel processes them before an already served queue. We will use FQ-CoDel in the last part of Chapter 3, which follows next.

3

Deployable Transport Optimizations

In this chapter, we tackle our first research question, namely, *what is the impact of Internet giants on Internet transport evolution?*

As we have explored in Section 2.1, the Internet is considered ossified around the initial design of transport protocols. Still, transport protocols significantly impact the performance of Internet communication and specifically the Web.

Especially the Transmission Control Protocol (TCP), as the foundation of modern transport, is known to be extremely difficult to extend in a deployable manner. For example, Google initially proposed TCP Fast Open in 2011 [RCC+11] and the Internet Engineering Task Force (IETF) standardized it in 2014 [RFC7413] in hopes of offering reduced connection setup times through a cookie mechanism. Figure 3.1 shows the support for TCP Fast Open in Internet Protocol Version 4 (IPv4) on port 80, which we measure every week by recording hosts that respond to the Fast Open option. The figure shows that, initially the overall support was minimal but grew steadily over time. Nevertheless, a significant concern is the number of Internet Protocol (IP) addresses that answer with an invalid cookie which is likely stripped by middleboxes, or some hosts implement the option improperly. While the relative number of IP addresses has significantly gone down, as of August 2019, there are still over 76k IP addresses that cause an invalid cookie response. We did not check whether or not requesting the option will later hinder the communication with those hosts. However, a practical example with the RWTH Aachen University's firewall showed that while we are successful in negotiating the option, later communication fails as the SYN/ACK packet may already acknowledge data, and thus its acknowledgment (ACK) number increases by more than the SYN, causing our firewall to drop the packet and stalling the communication. Of course, other firewalls may show similar but also other flaws.

To this end, many optimizations to TCP today are end-host only optimizations, meaning one can apply them to one end of the connection only without requiring

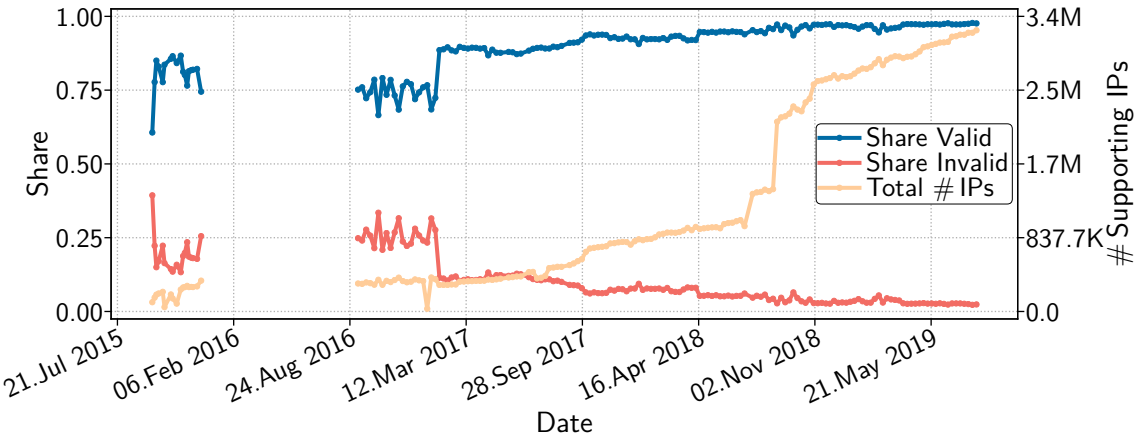


Figure 3.1 TCP Fast Open support in IPv4 on port 80. Please note that there were no scans from January 2016 to August 2016.

changes to the other party. These changes are often parameters of well-known functionality or changes to algorithms that TCP uses internally. To this end, a recent proposal named recent ACK (RACK) [CCD+19] departs from traditional packet or sequence counting for loss-recovery and solely used the notion of time to detect loss. It is sufficient to implement RACK on the sender and thus does not require changing the wire image or the receiver. A similar, heavily discussed parameter of a TCP connection is TCP's initial congestion window (IW), which is the topic of our first contribution (Section 3.1) in this chapter. Similar to RACK, it is a sender-only change that, however, significantly impacts the performance of short Web flows. We chose this particular feature to investigate how TCP has adjusted to the changes in infrastructures and connectivity. Within our first contribution, we strive to understand how IWs are used in the Internet at large, i.e., in all of IPv4 as well as specifically by Internet giants which we suspect to fine-tune this parameter. Our measurements enable us to understand how Internet practice evolved and whether or not Internet reality matches standardized practices.

Second, we shift our focus to a radical redesign of the transport layer with QUIC which is currently pushed forward by some Internet giants. By focussing on QUIC, we are able to judge how new developments affect Internet operation and thereby complement our previous investigations of TCP. Traditionally, TCP headers have been used to measure loss, reordering, and the round-trip time (RTT) of flows in a network. QUIC is said to significantly challenge the operation of (mobile) networks as it is a fully encrypted transport hiding all relevant headers from a passive observer. In Section 3.2, we design measurements to investigate the birth, global deployment, and key players of this new Internet protocol. We further analyze QUIC's traffic share using several network traces to investigate who uses QUIC and whether QUIC already challenges network operators today. Subsequently, we shine a light on whether QUIC offers the promised performance increases in comparison to TCP and whether humans are able to perceive these differences. Fueled by our previous contribution, we put a special emphasis on whether a TCP parameterized in the way currently used by Internet giants, makes a difference when doing this comparison.

Finally, Section 3.3 regards the principal property of every transport protocol, i.e., congestion control (CC) and how it practically affects the resource sharing in today's Internet. We investigate how Internet giants operate their CC to shine a light on how single big players influence the Internet landscape and the principles on which the Internet operates. We further show how individual users can regain control in this highly distributed system to enable fair and low latency resource sharing. By focussing on CC, we bring the essence of the previous contributions together enabling us to judge beyond individual protocols.

These three contributions thus deliver one view to our initial, continuously unfolding, question how Internet giants evolve Internet transport. We, however, have to overcome several key research challenges (apart from designing the measurements themselves) in this chapter to answer this question which we highlight in the following.

Research Challenges

- **How are we able to perform transport protocol measurements at an Internet-scale?**

  As we want to grasp the deployment of transport protocol features in the Internet at large, we need to design our measurements in a way such that they scale to billions of systems in a timely manner. This becomes increasingly difficult if the features that are to be tested are features that can only be observed through behavior and are not explicitly negotiated.

- **How much scanning is enough?**

  As we have highlighted in Section 2.2.1, scanning may have an impact on the Internet and the networks that we are scanning. Specifically, if the measurements go beyond a simple one-time packet exchange, we need to investigate how we can reduce the impact of our scans while maintaining the value and expressiveness. This further enables to regularly probe the Internet for the features involved, and thus, allow us to witness its evolution on a small time-scale.

- **How do we measure the (human-perceivable) performance of a transport protocol?**

  With our Internet-wide measurements, we uncover how a transport is used, but we lack the insights into how the individual configurations actually affect the performance. To this end, we need to find ways to attest the performance of the transport protocols, especially comparing the different parameterizations. Further, while such performance measurements may show an advantage when looking at pure technical metrics, it is unclear whether or not these performance increases actually make a difference. For example, do website visitors have a different quality of experience (QoE) when using the different transports?

- **How can we measure Internet-scale distributed algorithms in a controlled and comparable setting?**

As we have discussed in Section 2.3.3.5, CC fairness critically depends on RTT. Thus, to investigate how Internet giants impact CC, we must enable eye-level comparisons for actual Internet deployments.

We now dive into our first contribution and analyze how TCP's long-debated IW is practically configured in the Internet at large and by Internet giants in particular.

3.1 Small Change, Big Effect – TCP's Initial Congestion Window

The Internet and specifically the Web have transformed the way we gather information, interact, or do business. This increasing dependence on the Web has fueled a pursuit of researchers and operators to develop and implement Web performance optimizations. For example, Google has pushed several improvements to Web technology, including new protocols such as Hypertext Transfer Protocol (HTTP)/2 (through SPDY) or QUIC (see Section 3.2), both of which have found swift adoption [VSN+16, RPD+18, ZRW+17] by others. Apart from technological advances, content delivery networks (CDNs) have changed the Internet on an architectural scale (see Section 2.1). Their ongoing quest to serve Web content from nearby servers has flattened the hierarchical structure of the Internet [GAL+08, LIM+10], thereby promising lower latencies. In pursuit of performance, CDNs are known to be early adaptors of new technology in an attempt to optimize the Web experience for their customers.

While adopting new technologies offers promising gains, their correct configuration is often challenging — e.g., HTTP/2 server push was a highly anticipated feature but is now known to be notoriously hard to use [ZRW+17, ZWH17, ZWH+18]. Some of these configuration challenges stem from the fact that they are dependent on network and application characteristics.

One long-debated performance configuration parameter is TCP's (and QUIC's) initial congestion window (IW) size. The IW characterizes the performance at the beginning of a new connection and is a vital component of all CC algorithms. The IW size bootstraps the congestion window (cwnd) and thus controls the amount of unacknowledged data sent at connection start and thereby heavily influences the start-up behavior of new and particularly short-lived connections (e.g., typical Web transfers) or those that are revived from idle. A small IW can prolong transmissions and cause unnecessary latency as the transport protocol needs to await feedback (ACKs) to increase the cwnd. Contrary, too large IWs can lead to loss and retransmissions when the network cannot handle large bursts of data. Thus choosing the optimal value for each network is critical for high performance — and thus interesting for CDNs.

Despite its relevance, the IW size is typically regarded as a *static* parameter whose IETF-recommended size should fit all networks and applications. Since its first definition to one segment in 1988 [Jac88], its recommended size has only changed twice, to two to four segments in 1998 [RFC2414, RFC3390] and — motivated by increasing access speeds and its promise to shorten page loading times [DRC+10] — to ten segments in 2013 [RFC6928]. Flores et al. [FKB16] recently showed that IW customization helps to reduce CDN latency. In this regard, a small-scale study by CDNPlanet showed that half of the probed CDNs use IW10 as the IETF-recommended size, while others already use larger IW sizes [CDN17]. Others, e.g., [All15], even argue to abandon static IETF-standardized values for the IW to enable customization already in the standards. Nonetheless, little is known about IW configurations, especially in case of Internet giants such as CDNs.

In this contribution, we broadly probe IPv4 hosts and specifically CDNs to gather an empirical understanding of how IW customization already takes place in today's Internet. We investigate the prevalence of IETF-recommended values in samples of IPv4 for over three years. Further, we investigate CDN IW configurations from globally distributed vantage points and from our University's network, thereby shedding light on the degree of customization that CDNs show today. Our results show that the Internet at large converges towards the current IETF recommendation of ten segments. On the other hand, we observe that IW customization beyond standardized practices is already common practice, and there exists a gap between standardization, research, and Internet reality.

Driven by this gap, we further investigate the effects of this new Internet reality on the performance of slow start when competing against other flows. To this end, we seed a testbed study by our real-world observations and investigate the start-up performance benefits and disadvantages when the traditional CUBIC (see 2.3.3.2) and the recent bottleneck bandwidth and round-trip propagation time (BBR) (see Section 2.3.3.3) CC have to compete for traffic against an elephant flow[10]. Our findings indicate that the way CDNs utilize IW customization can indeed yield drastic performance increases. Specifically, this section contributes the following:

- The first long term analysis of the evolution of IW configurations in IPv4. Our results show a convergence towards the IETF-recommended values at Internet-scale.
- We provide a comprehensive analysis of current IW configuration practices of CDNs. We show that IWs are configured up to ten times larger than recommended by IETF's current experimental standard.
- Further, we find multiple CDNs which use customized IW configurations, i.e., deliver data using different IWs for different customers or service types. We observe that larger IWs are, for example, preferred for streamed video instances, yet, content types do not necessarily enforce specific IW settings.
- By analyzing IWs through different geographically distributed networks, we find instances of network-dependent IW configurations of CDNs.
- We investigate the burstiness of IWs and find that some CDNs utilize pacing to space out packets over time during slow start, potentially reducing the chance of losses.
- Our measurements show different configurations of packet pacing, challenging the traditional notion of IWs, we find that a data rate better captures the demand on a network than a fixed number of segments.
- We build a testbed to evaluate our real-world observations in a controlled environment; we find that increasing IWs *can* increase or hurt performance, specifically, our investigations show that increasing the IW should go hand in hand with TCP pacing to actually benefit.

Structure. The remainder of this contribution is structured as follows. Section 3.1.1 discusses related work and shows how IWs are defined, standardized, and how they impact performance. Section 3.1.2 introduces our IW scanning methodology, IP-address-based IPv4 scans, and the resulting changes to our scanner architecture to

[10] An elephant flow, in contrast to a mice flow, is a *long-running* flow that seeks to maximize its bandwidth

investigate CDNs. Following, Section 3.1.4 to Section 3.1.6 paint the global CDN IW configuration space by discussing CDN specific IW configuration. In Section 3.1.7, we project our real-world findings to a testbed and investigate its impact on flow start-up behavior. Finally, Section 3.1.8 concludes our findings.

3.1.1 TCP's Initial Congestion Window

We start by exploring TCP's IW: *i)* how it is defined and sized, *ii)* how its size influences flow completion time (FCT), and *iii)* how related works have gathered an understanding on IWs through Internet measurements.

IW Definition. TCP's IW governs the number of unacknowledged bytes in flight until the first acknowledgment is received. That is typically data sent in the first roundtrip of a connection after TCP has completed the three-way handshake. Thus, the IW at the sender-side and the receive window at the receiver-side define the application's data rate at the start of the connection and bootstraps the window doubling during slow start. Furthermore, depending on the CC algorithm, the IW is also used after long idle periods for restart (e.g., in Web browsers when using HTTP/2).

IW Size Definition. The IW is typically defined in bytes and often operating systems allow configuration of the IW in multiples of the maximum segment size (MSS). To this end, many Requests for Comments (RFCs) (here at the example of [RFC6928]) define a dualism for the IW, either in terms of the multiple of the MSS, e.g., IW10 for ten segments worth of data, or by an upper limit of bytes, e.g., 14 600 B typically corresponding to a classic full ethernet frame minus TCP and IP headers times the multiple.

3.1.1.1 Testbed Study: Impact of IW Size on Internet Performance

To highlight the impact of different IW sizes on Internet performance, we conduct a testbed study. The testbed involves two directly connected Gigabit Ethernet Linux hosts whose link bandwidth and latency are controlled by NetEm in each host. We select four bandwidth configurations (i.e., 4, 7, 26, and 100 Mbit/s) and three delay configurations (i.e., 30, 100, and 250 ms) to reflect typical Internet access characteristics reported by Akamai [Aka16]. We further choose six IWs: 4, 10, 16, 20, 32, and 50 segments, to reflect the current standard of 4 segments [RFC3390], the current IETF recommendation of ten segments [RFC6928], and larger IWs. In each experiment, we transfer a single flow of size 71 kB, i.e., 50 frames of data (the average size of the Google landing page in 2017). For each configuration, we repeat each experiment 30 times.

Flow Completion Time. Given its relevance to Web browsing, we first measure the TCP flows' average FCTs (AFCTs) subject to the different parameters (i.e., IW size, RTT, and bandwidth). We define the AFCT as the average time to the last byte of the flow. Figure 3.2 shows the AFCT and its standard deviation for the different parameters. It shows that increasing the IW reduces the AFCT if the link speed or

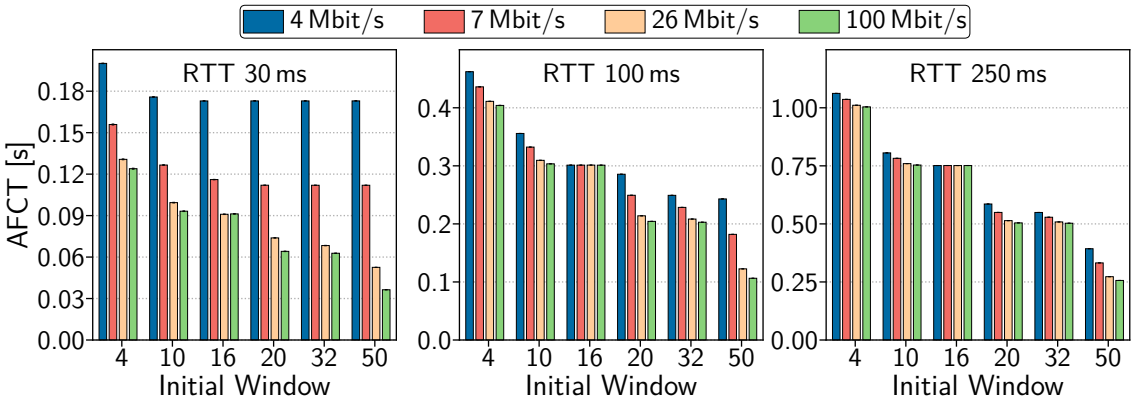


Figure 3.2 AFCT when varying bandwidth, latency, and IWs. Horizontal lines mark roundtrips. Larger IWs *can* improve the latency.

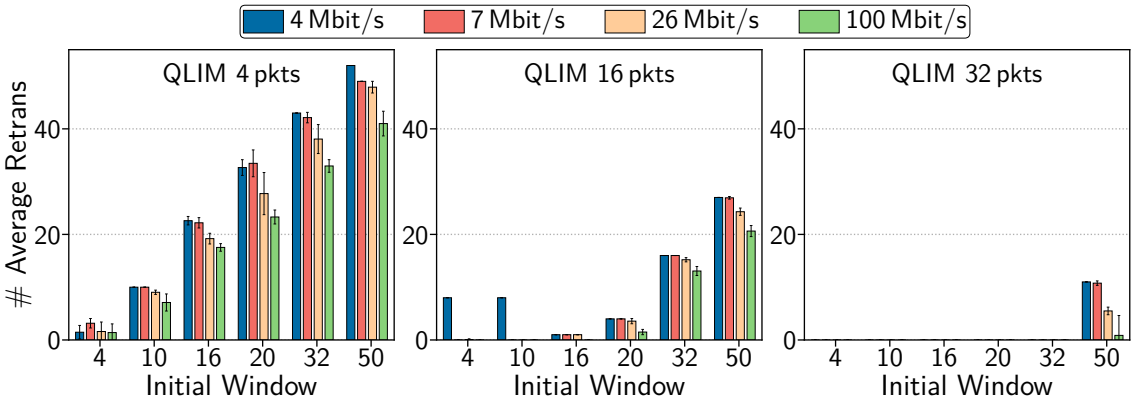


Figure 3.3 Losses increase when increasing IWs depending on bottleneck bandwidth and queue sizes. Benefits of increased IWs are highly network-dependent.

RTT is sufficiently high. For low bandwidth connections with low latency, larger IWs have effectively no impact on the AFCT as these connections are limited by throughput. However, when higher speeds are available, increased IWs can effectively shorten the required roundtrips to finish the data transfer — a key motivation for CDNs to configure larger IWs.

Retransmissions. Large IWs, however, yield more bursty traffic that can lead to temporary phases of congestion more easily, reflected in higher loss rates. To highlight this effect, we conduct a second experiment which measures the average retransmission rates of the TCP flows subject to different bottleneck link configurations. We realize this setting by now connecting the hosts via a bottleneck router with different bandwidth capacities and queue sizes (QLIMs) of a regular drop tail FIFO queue. We again transfer 71 kB and vary the IW configurations for each bandwidth, queue size, and IW triple, testing every configuration 30 times. We show the average retransmissions required and standard deviation in Figure 3.3.

As the figures show, increasing the IW can have detrimental effects on the connection. We observe that larger IWs cause higher loss rates when either the bottleneck bandwidths or the bottleneck queue sizes are too small. Considering the retransmissions for the smallest queue size, we can see that large IWs cause heavy losses. Increasing

the queue size helps in buffering the IWs, yet at the cost of added latency, e.g., a 7 Mbit/s link with a buffer of 16 packets can add up to 27 ms of delay to a packet. Thus, simply increasing the queue size is not the desired solution as we have already discussed in Section 2.3.4. While these motivating measurements neglect multiple users sharing the bottleneck, loss-based congestion control of multiple users will lead to full queues all of the time leading to tight buffer space for new flows as shown in these measurements.

Takeaway. *Our study shows, similar to related works [DRC+10, Sch09] that the IW size can strongly influence flow performance but can also overload congested or low-bandwidth connections. It is thus key to CC to correctly set an IW that adequately balances throughput and loss to bootstrap a TCP connection.*

3.1.1.2 Related Work

The relevance of TCP's IW size is reflected in extensive debates and a successive evolution of its value in the TCP standards over the last decades. Initially, in 1988, Jacobson [Jac88] set the IW to one segment, and nine years later, [RFC2001] standardized it. [RFC2414] experimentally extended this setting to two to four segments (or 4380 B) in 1998 and [RFC3390] later made it a proposed standard — a setting that remained untouched for a decade. Motivated by the increase of network access speeds and the desire to reduce Web page loading times, [DRC+10] proposed in 2010 and later [RFC6928] recommended to increase the IW to ten segments in 2013. Most recently, Allman [All15] even argues for abandoning a specification of the IW size and thus arguing to end a decades-long debate about its precise value in the standards. This argument is motivated by allowing hosts to configure more tailored IWs.

Given the relevance of the IW on both flow completion times and Internet traffic burstiness leading to losses, an empirical study of the IW is necessary to understand current network performance aspects. This understanding has been gained in both active and passive measurement studies. Concerning active measurements, Medina et al. [MAF05] probed 85 k servers in 2004 and found most servers to be on an IW of one or two with only 1% of hosts having an IW larger than four. Our measurements are similar to those of Medina et al. in terms of methodology. In contrast, we specifically target CDNs, which were still on the rise in 2004 and did not have as much of a footprint as they have today. Regarding passive measurements, Qian et al. [QGM+09] inferred IW distributions from several traces in 2009. While their dataset covers traces captured in a diverse set of networks and also covers non-publicly visible hosts, they did not discuss the impact of CDNs in their study. A small-scale study by CDNPlanet [CDN17] probed 15 CDNs via HTTP and found six to use IW10 and others to use larger IWs. Our work is similar to that of CDNPlanet, we share the same goal to shed light on CDN IWs, but their methodology (manual inspection of packet traces) limits a broad assessment of CDN IW configurations which is one focus of this contribution. An unknown vantage point and limiting TCP receive windows further limits the comparability of their study to ours.

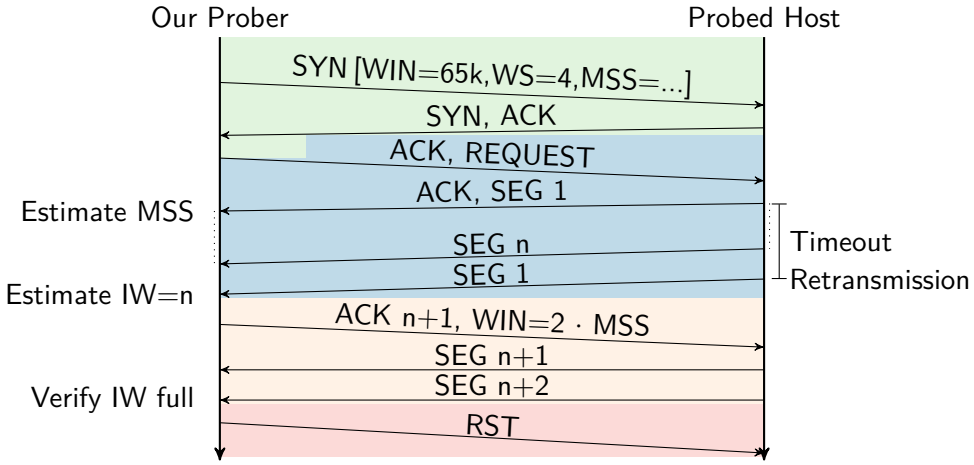


Figure 3.4 Scan procedure: MSS and large receive window are announced and no ACKs are sent until a retransmission. The estimated IW is the # bytes received before the retransmission.

The idea of pacing TCP flows (see Section 2.3.3.4) goes back to the observations of ACK clocking in [Jac88] which motivated pacing in [VH97] to clock data until ACKs arrive, but Aggarwal et al. [ASA00] were the first to study it extensively. Their simulative results employ a non-bursty pacing implementation, and their focus is on long-lived connections in contrast to the impact of pacing at the start of the connection. Similarly, work on router buffer sizing [EGG+06] has shown that tiny buffers can only be realized when some form of pacing is used (see Section 2.3.4), an observation that we qualitatively validate. Wei et al. [WCL06] replicate many experiments from [ASA00], e.g., with different TCP variants, and their simulations show that the CC algorithm itself has a tremendous impact. Motivated by this, we rely on the emulation of real implementations to study how pacing affects the slow start behavior, in this regard TCP Jump Start [LAJ+07] is similar by abandoning an IW and simply pacing out all data. Again, this work bases on simulations, to the best of our knowledge, the behavior of Linux's pacer has not been analyzed in academic work, which we contribute.

3.1.2 Measuring IWs

We begin by summarizing our IW size estimation approach. To also enable measuring CDNs, we will then extend our scheme to also account for virtualization by incorporating per-CDN target uniform resource locators (URLs) and hostnames. This specialization is needed to fetch large content from CDNs for IW estimation. We next describe its general procedure and details that we modify to account for CDN properties.

For explanation, we split the IW estimation procedure (visualized in Figure 3.4) into four phases: *First*, we perform a regular TCP handshake announcing the largest possible receive window (rwnd) of 65 kB and, to account for overly large IWs configured at some CDNs, also a window scaling option (shifting the window by three bits) to never block the data transmission due to flow control. Since IWs can be configured depending on the MSS, we additionally set an MSS (varied by later

measurements). No further options like Selective Acknowledgements, which can, e.g., cause TCP tail-loss probes that would challenge IW estimation, are activated.

After establishing the TCP connection, the *second* phase starts by transmitting a request in hope to trigger a response that exceeds the configured IW at the probed host. The probed host will now commence sending the requested resource. However, we are not going to generate acknowledgments for any segment that we receive, and thus, the cwnd does not increase from the IW, and the host can only send as many bytes as the IW.

By not acknowledging segments, the probed host will eventually initiate a retransmission of the first, from its point of view, lost segment, which heralds the start of the *third* phase. Either, the IW limited the sending host, or it ran out of data to send. To test for this, we start acknowledging the last segment enabling the host to continue sending data, and if the host does so, we know that the host did not run out of data. At this point, we can estimate the IW by observing the sequence number space and segment sizes that we received before the retransmission.

Finally, the *last* phase consists of tearing down the connection with TCP's RST mechanism. As this IW estimation methodology fails when tail-loss occurs (i.e., loss of the last packet in IW), we recommend performing multiple scans of the same host.

Implementation & Validation. We implement our approach in go-lang (source available at [Rüt18c]) to benefit from its multiprocessing capabilities and also in ZMap [DWH13] (source available at [Rüt17]). To test both implementations and to validate the correctness of the IW estimation, we run them in a Mininet [LHM10]. We use iptable's statistic module to drop packets at the head, within, and at the tail of the IW to validate the estimation correctness, i.e., correct estimations for the first two, and a reduced IW for the latter case (tail-loss). Further, we vary the IW size and available bytes on the server-side and the announced MSS at the probing client to validate non-standard IWs and out-of-data situations in various settings. Our tools always estimated the IW correctly except for tail-loss (as expected).

3.1.3 Measuring IW Configurations in the Wild

We start exploring TCP IW configurations in IPv4 using an implementation based on ZMap. Looking at all of IPv4 allows painting a holistic picture of the IW deployment world-wide. However, as we will see, it cannot test CDNs properly. Our ZMap tool builds on top of two request models that are motivated by their large scale availability in the Internet, i.e., HTTP and Transport Layer Security (TLS).

3.1.3.1 HTTP-based IW Inference

Our methodology relies (see Figure 3.4) on performing a request that should trigger a large transmission from the probed target. To generate sufficient response data, we base our first IW inference method on probing HTTP servers. Our motivation lies in the widespread deployment of HTTP as the major application layer protocol,

which thus provides a strong candidate for our IW scans. According to [ACF+12], HTTP accounts for over 50% of traffic at a major European Internet exchange point (IXP) and, according to our scans, we can successfully exchange data with ≈ 48.3M hosts on port 80. For the same reasons, HTTP was used in prior works to infer the IW size, e.g., Padhye and Floyd [PF01] or Medina et al. [MAF05]. Both works can only ensure large enough responses that fill the IW by providing URL lists defining an appropriate request for each probed host. However, an extensive assessment of the entire IPv4 space is *not* feasible by relying on prior knowledge. Consequently, we propose an extended approach that allows inferring the IW of HTTP servers *without* any prior knowledge such as precompiled URL lists triggering large responses.

Our proposed approach is as follows; we initially request the root (`/`) page hoping that it contains enough payload to fill the IW. As we have no prior knowledge of the host, we can only include the IP address in the mandatory HTTP `Host` header (a shortcoming as we will see for measuring CDN). Many (virtualized) servers will reply with a `301 Moved Permanently` error, and thus, we can extract a valid uniform resource identifier (URI) from the `Location` header in the error response. In these cases, the extracted URI will redirect us to a valid page. We can further provide a valid `Host` header if the URI includes the host's common name. This information enables us to issue another request that hopefully results in a larger response. So, we send an `RST` to quickly end the connection and issue another request on a new connection following the redirect.

If the redirect fails, we increase the response size by bloating possible error pages. This approach is motivated by an initial observation that a substantial number of servers replies with `404 Not Found` pages that include the URI that they could not find. Thus, enlarging the request URI will enlarge the error response. Hence, we request a long URI indicating the nature of the scan, anticipating a long enough response. We choose the URI to fill up the maximum transmission unit (MTU) of our connection, thus transmitting more bytes than we announced we would be capable of in the MSS.

In addition to acknowledging segments to look for more data, we additionally infer if the IW was actually exhausted by exploiting HTTP characteristics. Concretely, we request that the remote-end closes the connection upon finishing the transmission by including the `Connection: close` HTTP header. This semantic should lead to a TCP `FIN` once the remote-end has transmitted all data. However, if the remote end has still data after it filled the IW, it cannot send the `FIN` as it still has data in its transmit queue. By receiving the `FIN`, we infer that the IW was not reached (in the absence of packet loss).

3.1.3.2 TLS-based IW Inference

Rising security and privacy concerns contributed to an increased usage of HTTP Secure (HTTPS) (the TLS tunneled version of HTTP) [NFL+14]. After port scans of the default HTTPS TCP port 443, we were able to exchange data with ≈ 42.6M hosts successfully. This share is further expected to grow. Not only traditional services (e.g., banking or e-commerce) are using TLS also Internet giants such as

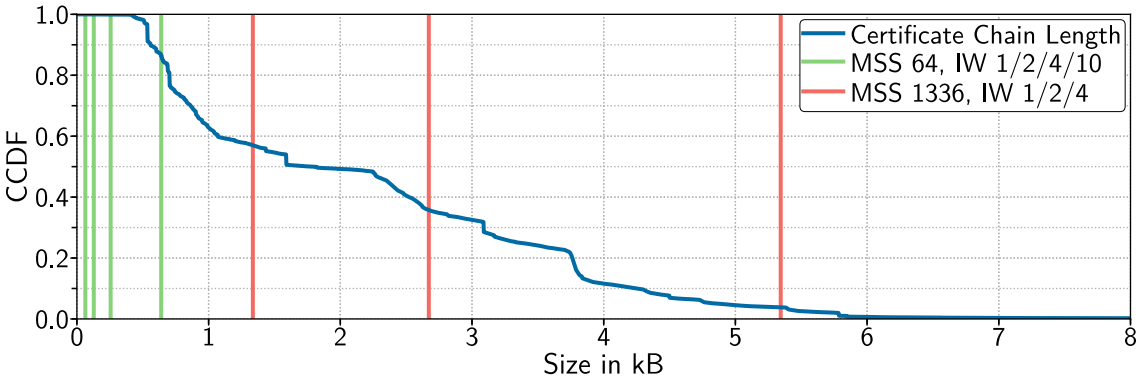


Figure 3.5 CCDF of certificate chain length of 36.5M hosts from censys.io data. TCP payload sizes covered with several IWs using MSS of 64 and a typical MSS of 1336 B[1].

Facebook or Google have started to secure all of their traffic, further motivating others to switch. This trend is also manifested in HTTP/2 [RFC7540] — even though not mandated by the standard, practically all current implementations enforce TLS. Given this increasing relevance, we next detail a TLS-based inference method.

This IW inference utilizes the TLS handshake in which the server sends a large response. In TLS, the handshake is initiated with a `client hello`, indicating, e.g., cipher suites or extensions. Upon reception, the server replies with a `server hello` choosing a cipher and depending on that, keying material. Most importantly, the server continues to transmit its certificate chain that is required to validate its trust. Certificates typically dominate the server's answer in the number of bytes.

We analyzed TLS handshakes using the data provided at censys.io. Figure 3.5 shows the complementary cumulative distribution function (CCDF) of the server certificate chain length of 36.5M hosts. On average, the certificate chain length was 2186 B (with a minimum of 36 B and a maximum of 65 kB). For our scan to succeed, the remote host needs to send us at least IW × MSS bytes. Assuming an MSS of 64 B and IW10, we only need 640 B of certificates which are supplied by more than 86% of the hosts. We can still reach 50% of the hosts even if they would use IW34. These calculations neglect the actual size of the `server hello` and possible extensions that follow, yielding even more payload to rely on.

To scan a host, we initiate the TLS handshake after completing the TCP handshake. Since completing the TLS handshake relies on the offered cipher suites by the client, we compiled a list of 40 TLS ciphers announced by Safari, Firefox, and Chrome and enriched the list with ciphers that we extracted from the censys.io data that were not already announced by the three browsers. To generate even more data, we included extensions for requesting OCSP stapling.

We rely on our acknowledging method to determine if the reply filled the IW or not. In contrast to HTTP, we could have inspected the TLS length fields and use these to determine if we can still expect more data to be available. However, looking

Scan	Reachable	Success	Few Data	Error
HTTP	48.3 M	50.8%	47.6%	1.6%
TLS	42.6 M	85.6%	13.3%	1.1%

Table 3.1 Scan data set overview (rounded) scanned with MSS 64. Reachable meaning data exchange is possible.

into payloads requires that we have no packet loss and it further complicates the implementation, and we found no advantage in doing so.

Implementation Challenges

ZMap is designed for a single packet exchange with the target to probe for open ports. Since this optimized port scan design is not capable of exchanging multiple packets with the target (as needed for valid TCP connections), we added this functionality in a lightweight fashion. We added a probe module to establish TCP connections and keep track of various per-connection properties such as the length of each segment and connection state. This design still allows us to perform fast scans, e.g., at a moderate scan rate of only 150k packets/s, our HTTP-based IW scan only needs 7.5 h to probe the entire IPv4 address space. An unmodified ZMap scanner performs a port scan at the same rate in only 6.8 h — recall that the unmodified scanner performs only a single packet exchange with the host, instead of full TCP connections with subsequent exchanges. This small difference highlights the efficiency of our scan method.

3.1.3.3 Results: IW Distributions in IPv4

Scan Setup. To evaluate the IW distribution on the Internet, we operate a scanner within our University's network. This operation is carefully coordinated with the University's IT Center to react to abuse e-mails adequately and to have unfiltered access to the Internet (e.g., without transparent Web proxies). We followed the guidelines in [DWH13] (see Section 2.2.1) and set up reverse Domain Name System (rDNS) entries and a Web page explaining the nature of the scans together with an opt-out mechanism. To this end, we do not scan unroutable or blacklisted IP addresses.

Dataset. We present results based on two scans performed in the second and third week of August 2017, which we summarize in Table 3.1. We declare a success, if we can estimate the IW, we mark a scan as *few data*, if we cannot be sure that the reply exhausted the IW, *error* marks all other cases (e.g., connection reset). For our measurements, we decided to probe each host three times to account for tail loss and count it successful if at least two out of three probes yield the same result

[1] We implemented a ZMap-based Internet Control Messaging Protocol (ICMP) path discovery following [RFC1191] estimating typical MSS values, highlighting the IW requirements of TLS. We found 99% (80%) of all hosts support an MSS of 1336 B (1436 B).

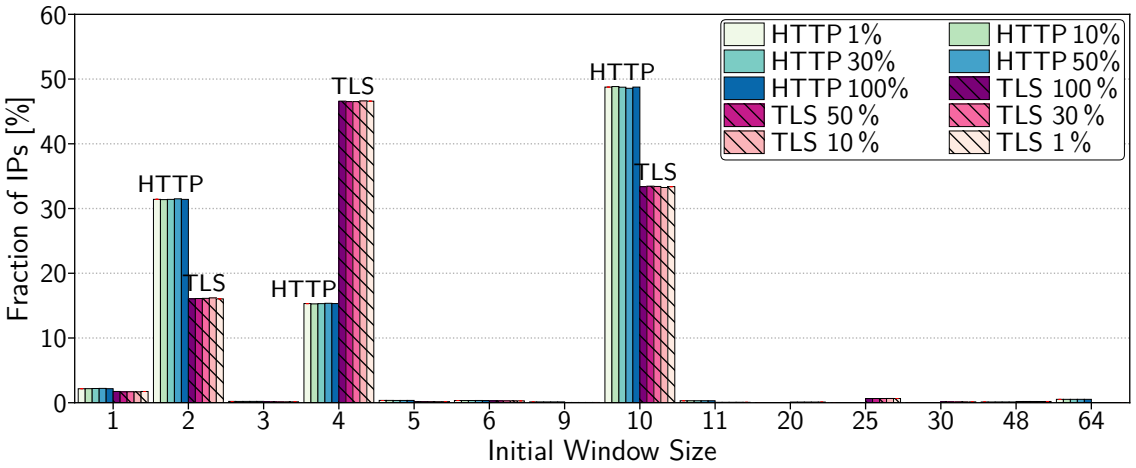


Figure 3.6 IW distribution in IPv4 of HTTP and TLS for IWs used by at least 0.1% of the hosts, probed with an MSS = 64 B.

and as tail loss may occur, we require them to be the maximum of all three probes. To further test if hosts adjust their IW based on the announced MSS (recall that the standard also defines a byte limit), i.e., to always transmit a certain number of bytes in contrast to segments, we scan with an MSS of 64 B and 128 B. To ensure no temporal changes at the host, we send all six probes (three for each MSS) after each other.

HTTP Scan Ethics. Our HTTP probing methodology is arguably more intrusive than TLS-based probing. The reason is that HTTP probing is requesting actual Web (error) pages and thus generates entries in server access logs. These entries triggered a significantly higher number of abuse e-mail than our TLS-based probing. As we will discuss in Section 3.1.3.4, the Internet-wide probing footprint can be drastically reduced by only probing a random sample of IP addresses to get stable IW distributions: currently, probing 1% of IPv4 suffices.

Success Rates. In total, we successfully probed 60.9M distinct IP addresses, of which 7M offered both services. Table 3.1 shows that TLS yields higher success rates than HTTP. HTTP probing of unknown hosts mainly suffers from not generating sufficient response data for IW inference. We tried to mitigate this by expanding error pages with long URLs, yet we found that, e.g., during our scans Akamai changed their default error page to not include the URL anymore highlighting the impact of our scans in server access logs. In contrast to HTTP, TLS returns more data (e.g., due to certificates) and is less intrusive. Still, around 13% of hosts return insufficient data. We attribute this to missing server name indication (SNI) (i.e., similar to HTTP's host-header a TLS extension that signifies the hostname) information since servers close connections when the request does not contain a (forward) Domain Name System (DNS) name, which are unavailable when only enumerating IP addresses.

3.1.3.4 Overall IW Distribution

We start by exploring the overall distribution of IW sizes for both HTTP and TLS and report results based on successful IW estimations (see Table 3.1) with an MSS of

Scan	NoData	IW1	IW2	IW3	IW4	IW5	IW6	IW7	IW8	IW9	IW10
HTTP	4.8%	16.5%	7.1%	7.2%	2.9%	3.6%	2.0%	45.0%	2.7%	1.1%	0.9%
TLS	17.8%	56.3%	5.6%	0.7%	1.9%	2.8%	2.4%	2.4%	3.4%	0.4%	0.8%

Table 3.2 Lower bounds of IWs for hosts that did not send enough data.

64 B. Figure 3.6 shows dominant IWs, i.e., observed at more than 0.1% of the hosts. We see that IWs of one, two, four, and ten segments dominate the scan. These IWs are present at more than 97% of all scanned HTTP or TLS hosts. This finding is in line with recommendations in various RFCs. Out of 7M IP addresses that appear for both HTTP and TLS, 6.2M agree in their IW estimate, and 858k IP addresses yield different IW estimates for HTTP and TLS. Interestingly, we find that the TLS scan and the HTTP scan differ in the distribution of IW4 and IW10: we find more TLS hosts with IW4. In contrast to the measurement by Medina et al. [MAF05] from 2005, we observe that IWs of four and ten segments have gained the highest relative growth. When analyzing non-standard IWs, we observe two peaks, one at 25 (TLS) and one at 64 segments (HTTP). However, the low overall deployment of IW10, especially on TLS-enabled host, is notable, given its standardization in [RFC6928] already in 2013 and its implementation, released in the Linux kernel 2.6.39 from May 2011, is even older.

Lower IW Bound for Hosts with Insufficient Data

As indicated by our success rate, we are roughly missing half of the HTTP (and 13.3% TLS) hosts by not having enough data available for IW probing (see "Few Data" in Table 3.1). To better understand these hosts, Table 3.2 shows their minimum supported IW, i.e., before they run out of data. The picture is different for HTTP and TLS. For HTTP, we find that 45.0% of probed hosts run out of data after having transmitted data worth of an IW of seven segments. Given the current standards, these hosts are likely actually configured to use an IW of ten segments. For TLS, 17.8% do not send any data (i.e., 4× more than for HTTP) and 56.3% run out of data after a single segment. This gap is likely caused by hosts not supporting our cipher suits or TLS versions offered by our probing module. Here, we are not receiving any certificates but only TLS error messages. In these cases, we cannot make speculations on possible IW configurations.

Scanning 1% is Enough!

During our scans, we have observed a significant number of abuse e-mails that are triggered by our scans, both automatic as well as manually created requests to us. To lower the footprint of our scans and to overcome one of our research challenges, we next investigate if we can reduce the footprint of our Internet-wide scans by limiting IW probing to a smaller subset of hosts. We thus extracted a random subset of 50%, 30% and 1% of all successfully probed IP addresses for both the HTTP and the TLS scan and show their IW distributions in Figure 3.6. For the 1% sample,

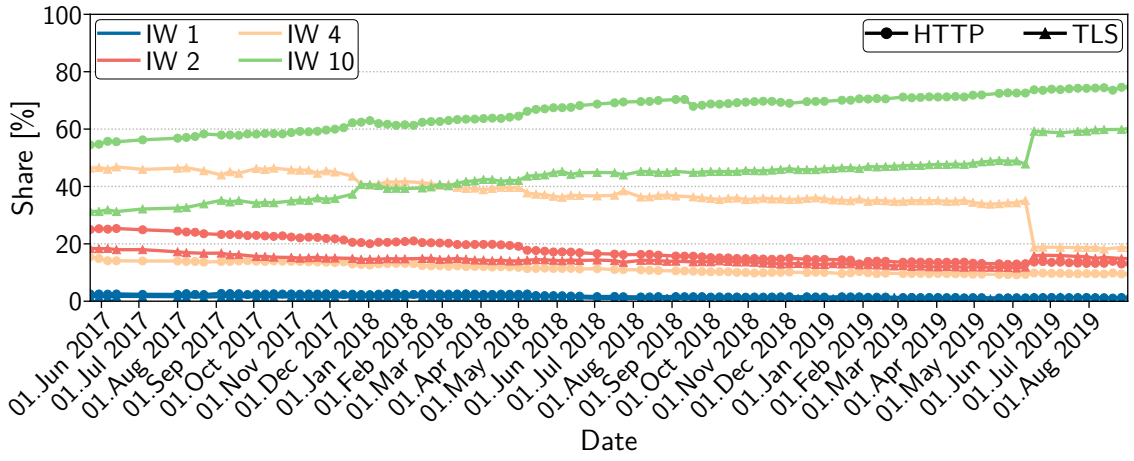


Figure 3.7 Evolution of IETF-recommended TCP IWs in IPv4 since June 2017 for HTTP (circles) and TLS (triangles) established through periodic 1% random samples.

we additionally show the mean of 30 random subsamples and the 99%-quantile in red (small and hardly visible in the figure). We observe a stable distribution for any sample size, indicating that only probing 1% of IPv4 already yields a stable distribution — even for IWs only present at 0.1% of the hosts. Since the first sample requires knowing the set of all IP addresses reachable for HTTP/TLS services, we further took 30 random 1% samples of the entire probable address space and arrived at the same result. While probing the entire IPv4 space is possible, it is (given current host configurations) not required to obtain representative IW estimates; reducing the overall footprint by only probing a random subset of 1% suffices.

Given our findings, we have been probing IPv4 weekly for over three years at random 1% samples for HTTP and TLS[11]. Figure 3.7 shows the evolution of IETF-recommended values. Throughout our observations, we see a steady increase in IW10 for both HTTP and TLS. For TLS, we observe that the share of IW10 starts to dominate that of IW4 in the first quarter of 2018. We suspect that operating system updates mainly drive the increase as current versions of Linux and Windows default to IW10.

Impact of Legacy Systems on Results

Since the overall distribution is likely to be impacted by (older) legacy systems, we next focus on assessing popular Internet infrastructures by scanning the Alexa Top 1M list. We show the IW distribution for the Alexa Top 1M list in Figure 3.8 (note the log-scale). In contrast to probing the entire IPv4 space, the success rate at popular hosts for HTTP increases to 80%, yet, TLS only gains marginally and succeeds at 85% of the hosts. We can now see that IW10, as the currently recommended value, dominates the scans with a support of over 85% (80%) for HTTP (TLS). Still, some hosts are on IW2 and IW4. The IW distribution of TLS hosts is irrespective of their Alexa rank, and only IW10 is more pronounced for higher-ranked HTTP hosts. We believe that in contrast to the entire IPv4 space, hosts of popular domains are

[11]Up to date data is provided at `https://iw.netray.io`

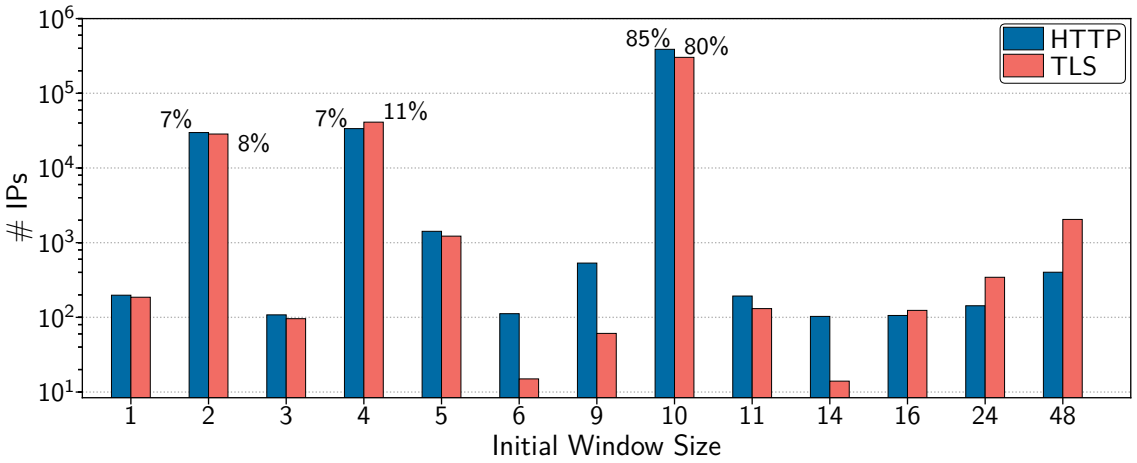


Figure 3.8 Alexa 1 M IW distribution of HTTP and TLS scan for IWs used by at least 100 hosts.

interested in performance optimizations or at least keep their systems up to date. Before we dig deeper into understanding these differences, we next discuss how hosts define their IW by looking at the differences when scanning with a larger MSS before we analyze the data on a per-autonomous system (AS)/service basis.

3.1.3.5 IW Defined by Byte Limit

Until now, we have only shown the results for our scan with an MSS of 64 B. We found that only a limited number (around 1%) of the scanned hosts adjusts their IW according to the announced MSS (when comparing with the results of the 128 B MSS scan). Roughly 50% of these hosts send 64 segments, and when doubling the MSS to 128 B, the segment number halves to 32. These findings suggest that these hosts are configured to use 4 kB as their IW, i.e., the product of the MSS and number of segments. We randomly sampled the hosts and manually investigated them to see if we can characterize the hosts. Eight out of ten hosts present a login interface to what seems to be residential access modems from Technicolor in different versions, and the Mexican Internet service provider (ISP) Telmex seems to host most of these modems. Among the others, we found publicly accessible power supply monitors that show the same behavior. We could not cluster the remaining 50% of the hosts into larger groups as above. One group that we found by randomly sampling are hosts that seem to adjust their IW in a way that they fill the network's MTU, i.e., with an MSS of 64 B they send 24 segments, and on 128 B, they send 12, summing up to 1536 B.

3.1.3.6 IW Distribution by Network & Service

We now analyze the IW usage by *network* type represented through ASes. Accordingly, we cluster our data by AS number (ASN) with similar IW distributions using DBSCAN (w.r.t. IW1, IW2, IW4, IW10, and other). The lefthand side of Figure 3.9 shows unusually large clusters with similar IW distributions that represent

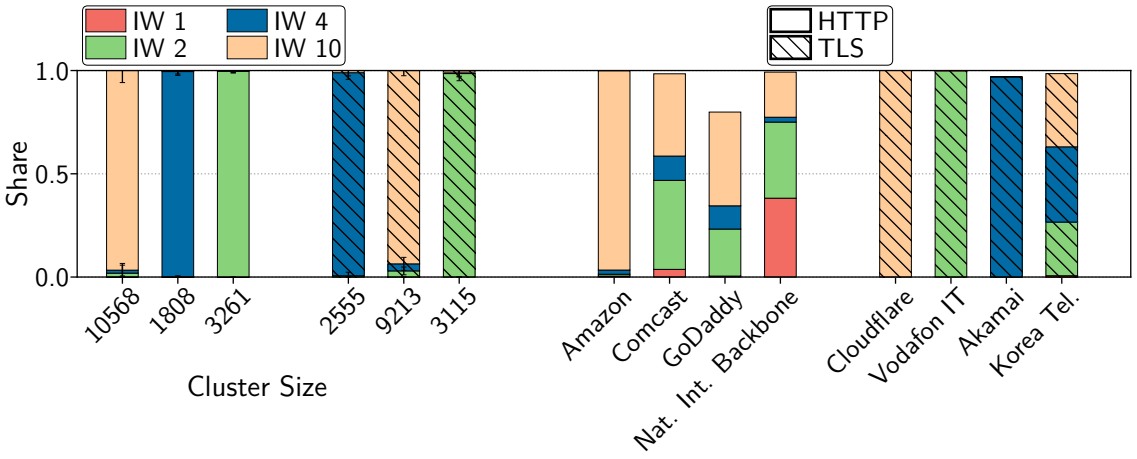


Figure 3.9 Distribution of IWs per AS. Left, 3 HTTP and 3 TLS clusters of ASes standing out. Right, representatives of these clusters or ASes that do not fit into the clusters.

a considerable fraction of all scanned IP addresses (HTTP 49%, TLS 48%). These clusters give intuition on per-service IW deployments. Clusters (HTTP and TLS) with nearly exclusive use of IW10 mostly compromise *content provider (CP)*, e.g., hosters, cloud provider, and CDNs. ASes with many IW2-based hosts belong to *ISPs* or in case of HTTP also to *universities*. The cluster for IW4 is a mixture between *ISPs* and *hosters*. While the HTTP measurement shows more ISPs than hosters, the TLS measurement stands out with an AS from Akamai that use IW4. In case of GoDaddy, 19.8% (32.7%) of the 137 k HTTP (193 k TLS) hosts that were announced by AS26496 (704 prefixes) use an IW of 48 segments. We remark that the number of GoDaddy hosts is $\ll 1\%$, which is why this IW peak is not clearly visible in Figure 3.6. Unlike our previous observed 4 kB IW hosts, these hosts use a static configuration of IW48, irrespective of the announced MSS. We found no apparent reason for this comparably large IW.

We find an assorted map of different IW configurations. To compare selected content and (residential) access networks, we show their IW distribution in Table 3.3. We classify content networks with the help of service-provider IP address ranges (e.g., [Ama19]) or the GHost HTTP server string in case of Akamai. Access networks are classified based on their rDNS record [SGS+17]: *i)* we extract hosts which encode their IP address in the rDNS record, i.e., 38.6% (62.5%) of all HTTP (TLS) IP addresses. To exclude server networks (e.g., Amazon and Akamai) we further match their rDNS record against a manually created ISP domain list and a keyword list (e.g., "customer", "dialin"). This way, we classify 16% (18.1%) of all HTTP (TLS) IP addresses as access. While it seems that CPs have widely adopted IW10, we still observe older IW configurations for networks with a potentially high share of legacy devices (e.g., home routers in access networks).

Besides differences between network types, content networks enable further per-service or even per-customer IW configuration (e.g., by Akamai [Aka15]). However, enumerating IP addresses is just not enough to grasp such service-dependent configurations as CDNs typically use the DNS name for exact content routing (see

	HTTP				TLS			
Service	IW1	IW2	IW4	IW10	IW1	IW2	IW4	IW10
Akamai	–	–	–	–	0.0	0.0	**100.0**	0.0
EC2	0.0	1.8	3.4	**94.7**	0.2	1.3	2.6	**95.8**
Cloudflare	0.0	0.0	0.0	**100.0**	0.0	0.0	0.0	**100.0**
Azure	0.0	7.8	**54.9**	37.1	0.1	4.1	**73.3**	21.9
Access NW	3.5	**50.2**	20.8	21.7	4.5	17.6	**67.1**	10.4

Table 3.3 Per-service IW distribution [%] clustered by IP address range (servers) or rDNS (access). Dominant IWs highlighted.

Section 2.1.2). This finding motivates an in-depth analysis of CPs, which is our next focus.

3.1.3.7 Measuring CDN IWs

To dissect the usage of IWs at CDNs, we must alter our scan setup. To this end, we structure our measurement study into three phases. At first, we gather lists of target URLs that are served by CDNs. In a second phase, we derive the IWs of the hosts serving the URLs. At last, we use virtual private networks (VPNs) to derive the IWs from different networks for a subset of these URLs.

Target Addresses

The first phase is relatively straightforward. Here, we utilize data published by the HTTPArchive [SGM+19]. The HTTPArchive crawls websites while recording diverse information about the websites. For their bi-monthly crawls, they visit all websites included in the Alexa Top 1M list. We utilize the crawl data from the 15th of January 2018 and extract all URLs that the browser loads during the crawl, i.e., the landing page URLs as well as subsequently requested objects such as images or JavaScripts. Even though the HTTPArchive already marks CDNs in their data, we repeat this step as the CDN choice could be geo-location dependent or utilize a MetaCDN (see Section 4.2) and as the HTTPArchive data can be up to half a month old, the CDN operator could have changed in the meantime. To do so, we apply the domain list [Goo18] published by the WebPagetest [AG18] framework (the framework driving the HTTPArchive), which enables to classify a URL by resolving its domain using DNS. Many CDNs utilize the DNS to redirect (using canonical name (CNAME) records) a user to the CDN server (see Section 2.1.2 for a detailed description). Thus a CDN can be identified by its CNAME pattern in the DNS resolution step, and for those which do not use CNAMEs, we utilize IP address lists published by the CPs, e.g., for Cloudflare. The result is a rather extensive list of URLs which we filter to include only URLs hosted at CDNs and only one URL per domain. For each domain, we choose the URL with the largest object size. This filtering results in a list of $\approx$ 227k URLs (available at [Rüt18c]) hosted on 69 CDNs

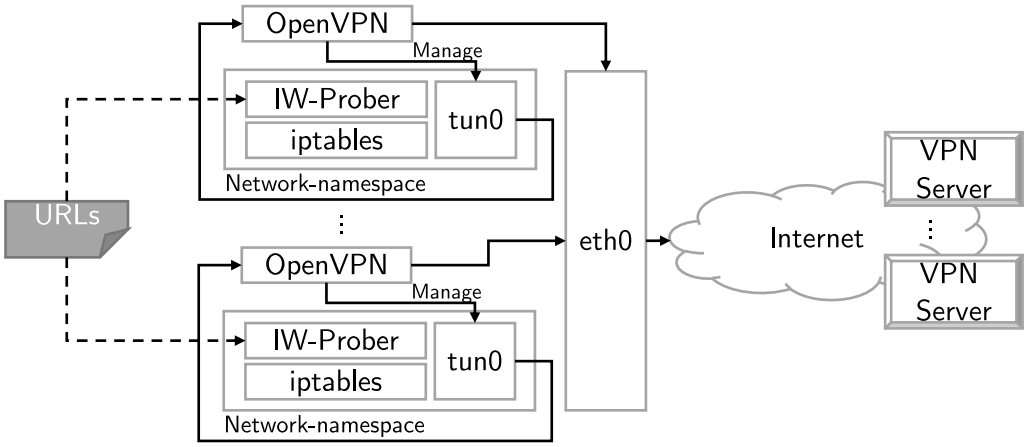


Figure 3.10 Overview of our scanning architecture. We leverage Linux network namespaces to scan concurrently and to easily manage multiple VPN connections.

used to establish IWs. We remove 116K objects (25 CDNs) that are too small to reliably estimate an IW (for large segment sizes, see Section 3.1.4.1) from the results.

Scanning Architecture

We use our go-lang-based scanner embedded in the architecture depicted in Figure 3.10 to structure and perform our scans. To enable concurrent scanning at multiple vantage points, we make use of OpenVPN and Linux's network namespaces. A network namespace can be seen as a shallow copy of the network stack with its own interfaces and routing tables. As many VPNs apply network address translation (NAT) to assign IP addresses to their peers, we experienced that different VPNs assigned the same IP address or the same subnet to us. To overcome this issue, we override OpenVPN's device creation and insert a script to manually create network devices in a new network namespace identified by the VPN's publicly facing IP address. This isolation enables to completely disregard any routing or name clash issues when using multiple VPNs in parallel. We then start one instance of an IW-prober in each namespace and feed it the URLs.

To not put an enormous burden on the VPNs, we perform a preprocessing step. Instead of querying all 111k URLs (potentially multiple times to account for tail losses) through the VPNs, we first derive a list of candidate URLs in our campus network. We select query candidates by grouping URLs hosted at the same CDN using the same IW and select a random sample of URLs for each (CDN, IW) pair.

Vantage Points

Gathering globally distributed vantage points that grant packet-level access is hard. To do so for our measurements, we make use of the VPN Gate [NS14] project by the University of Tsukuba. This project's goal is to give access to the Internet without censorship. To this end, the project manages a list of thousands of relay VPN servers around the globe. Volunteers operate many of them via their private

Internet uplinks. While the site lists many VPNs, we found only a small set of them to reliably work for our measurements, which might also be due to the implemented censorship protection announcing false gateway servers. To account for our scan methodology, we only use VPN connections made through TCP. Thus, loss between our VPN client and the VPN server is automatically resolved and does not affect our prober. According to [NS14], most of the VPN servers only have a relatively small bandwidth capacity mostly below 10 Mbit/s. Consequently, to not disturb the regular VPN operation, we implement a rate shaper into our prober that smooths burst and limits the outgoing bandwidth. We configure it to transmit at most 100 packets per second, and thus, we limit the prober to ≈1.2 Mbit/s for full-sized frames and much less for smaller frames. Further, through local experiments, we found that excessively parallelizing IW estimations challenge NATs quickly causing exhausted NAT tables. Therefore, we limit ourselves to a handful of parallel estimations per VPN. We chose both, the parallelism and the rate shaper, such that only short bursts are shaped and no long-standing queues are formed that could impact our measurements.

3.1.4 Campus Network Perspective on CDN IWs

We next explore CDN IW configurations from the perspective of a well-provisioned campus network (RWTH Aachen University) (worldwide perspective follows in Section 3.1.5) to set an upper bound on the expected IW sizes. As our network's upstream ISP peers at DE-CIX (where many CDNs peer as well), and our network offers at least one order of magnitude higher capacity than typical consumer Internet connections, CDNs could potentially adapt by serving content with higher IWs thus providing an upper bound on the expected IW sizes.

IW Probe Procedure

As IWs can be configured in bytes or segments, we scan each URL (see Section 3.1.3.7) with different maximum segment sizes of 64, 128, 536, 1200 B (leaving enough space for the VPN tunnel headers), ten times each. These repetition and different segment sizes enables to derive if the scanned host changes the total number of bytes delivered in the IW, i.e., the IW is fixed to a certain number of packets (we refer to the segments) or if it is fixed to a certain number of bytes (we refer to the bytes). To account for tail-loss, we perform a majority vote for each segment size and regard a scan as successful if >50% of the votes agree on the largest observed IW (97% of our measurements). To derive the final IW, we inspect the number of packets and bytes received over the four different segment sizes: if the IW depends on the segment size, we calculate an IW (in bytes) as if we were using maximum-sized segments (1460 B). Otherwise, we directly use the fixed number of bytes. Note that we refrain from showing quantities in which we observed certain IWs as our choice of URLs could bias them. Furthermore, we are not able to estimate IWs for all URLs, since their object size can be too small to fill a larger IW, which would bias the results towards smaller IWs.

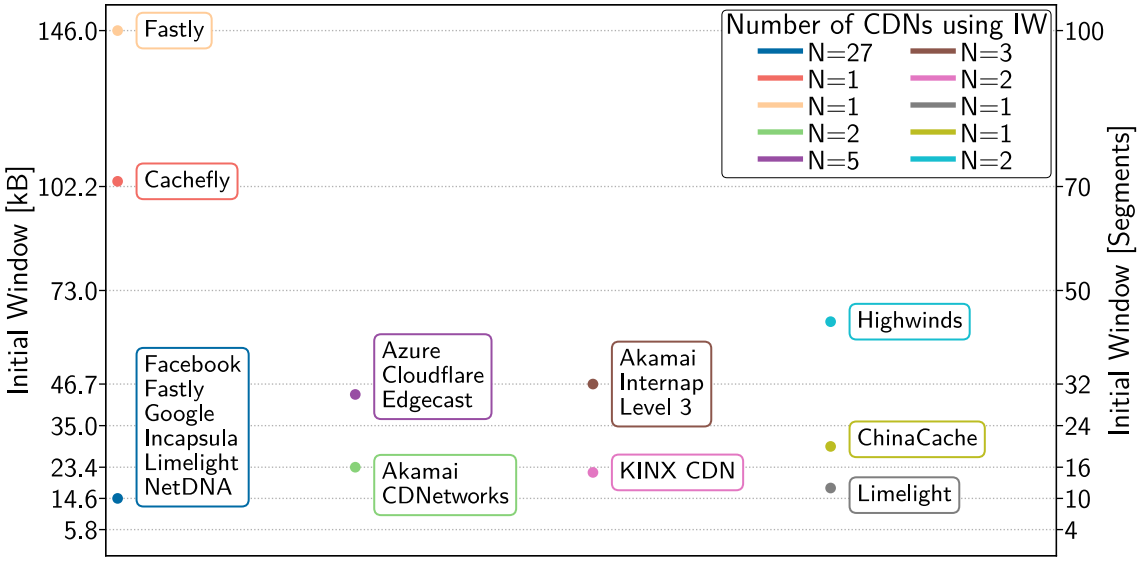


Figure 3.11 CDN IWs as seen from our university network. IETF sizes of four (not shown) and ten segments are present but also larger IWs.

3.1.4.1 IW Sizes

Figure 3.11 shows the resulting IW sizes in bytes and segments assuming 1460 B packets from our local campus network. Each dot represents an IW configuration, and the adjacent box lists a selection of CDN providers that deliver URLs with this IW (a CDN can occur in multiple boxes). Even though we find many CDNs offering URLs via IW10, we also find much larger sizes. This observation is in contrast to our prior IP address only scans over the IPv4 address space (see Section 3.1.3.3), which found IETF-recommended IW sizes to dominate most likely due to the number of deployed legacy systems (e.g., Digital Subscriber Line (DSL) gateways).

Our findings show that CDNs do, in fact, depart from IETF-recommended IW sizes and customize the IW. For example, we observe IW16 and IW32 for the probed Akamai URLs[12], both larger than the current IETF recommendation of IW10. However, we also find very large IWs. For example, the largest IW that we observed is by Fastly. They deliver *some* URLs using an IW of 100 segments. Cachefly also shows a larger than usual IW of 105 kB. Notably, Cachefly uses a fixed IW configured in bytes which leads to many transmitted segments when we scan using small segment sizes. On the opposite end of the spectrum, we find URLs hosted on CDNs that deliver data with a smaller IW than currently recommended. For example, we find URLs hosted on ChinaCache (not shown) that they deliver with an IW of four segments, yet, we can again observe that ChinaCache customizes as well, as they also deliver URLs with IW20.

[12]We remark that each CDN can use *additional* IW configurations beyond the configurations discovered in our measurements.

Operating System	rwnd [B]	WS	WIN [B]	Segs.
Linux 4.4	58	512	29 696	20
Android 6.0 (Linux 3.4)	685	128	87 680	60
Android 7.0 (Linux 3.18)	641	128	82 048	56
iOS 11.2.5	2058	64	131 712	90
Mac OS 10.9.5	8235	16	131 760	90
Mac OS 10.13.2	4117	32	131 744	90
Windows 7 (SP 1)	256	256	65 536	44
Windows 8.1 / 10	1024	256	262 144	179

Table 3.4 TCP rwnd, window scaling (WS), resulting window (WIN) in bytes and full-sized segments on different operating systems as reported on an HTTP GET request from an otherwise idling system.

Can Increased IWs be Utilized?

The actual amount of data that the CDNs transport is not only dependent on the server's cwnd. The client permanently announces a rwnd, TCP demands that no more than the minimum of the advertised rwnd and the cwnd is in flight. Table 3.4 shows the client-advertised receive window on an HTTP GET request for a selection of client operating systems. As the table highlights, the largest IWs that we measured would not be effective for a couple of operating systems. Linux 4.4 shows the lowest advertised rwnd, which would not be able to utilize many of our discovered CDN IWs. We found a git commit [DDM10] documenting this rwnd in response to the IW10 increase. Interestingly, Android, even though using an older Linux kernel, has increased the rwnd and would be able to utilize most of the IWs measured. The same holds for iOS and all other tested MacOS variants. Apart from Windows 7, all recent Windows variants announce rwnds large enough to not thwart even the largest observed IW.

Takeaway. *We observe CDNs to configure IW sizes beyond IETF-recommended values, highlighting that i) Internet reality departed from standardization and ii) IW sizes larger than standardized are of practical relevance. Their actual impact on network performance, in terms of losses, fairness and flow completion is practically unexplored by current research, highlighting that Internet reality also departed from research. We found that CDNs do customize IWs. However, it remains unclear when a CDN decides to utilize which IW.*

3.1.4.2 Are IWs Content-Dependent?

One way to customize IW sizes is by delivery service class (e.g., low latency Web delivery vs. elastic download), which can explain multiple observed IWs per CDN. Since we cannot directly identify service classes, we analyze IWs for typical *content types* by filtering the HTTPArchive for Akamai-served URLs according to their MIME type. We focus on Akamai, as one of the largest CDNs for which we already observed multiple IW sizes. For each domain, we take the largest URL of the following MIME types: *i)* `application/mp2t` (62 URLs) typically employed for video streaming

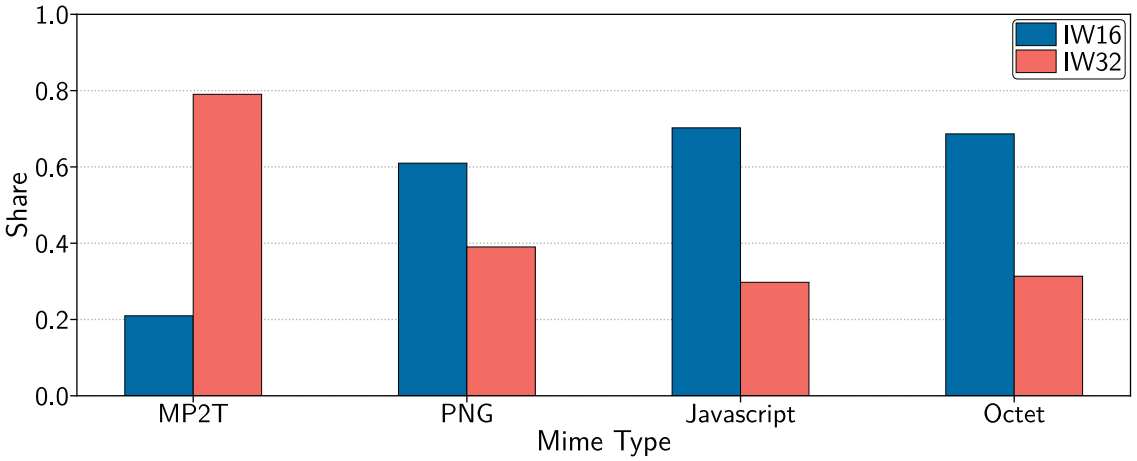


Figure 3.12 IW distribution for Akamai URLs per MIME type.

applications, *ii)* `image/png` (1812 URLs) for images, *iii)* `application/javascript` (1395 URLs) for regular Website content, and *iv)* `application/octet-stream` (67 URLs) for any binary data (download). We expect that interactive content uses the larger of the two IWs as, e.g., the play-out of a video should start as fast as possible, which large IWs would enable.

Figure 3.12 visualizes our analysis. Our expectations are partially met, i.e., streamed video content (MP2T) is mostly delivered with IW32, yet not exclusively. This fact and also the other MIME types highlight that the MIME type does not determine the IW per se. For PNGs, JavaScripts, and binary data, we observe that Akamai serves the majority via IW16; the quantity of IW32 varies between 30 % (JavaScript, binary) and 40 % (PNG). These observations highlight that it is more likely that an IW is not set depending on the MIME type but is rather dependent on the service class (product) that the customer has purchased at the CDN. Of course, some products are designed for interactive delivery and others not, yet, in the end, this non-strict assignment of IWs to MIME type shows that the customers decide what they deliver through which product.

Takeaway. *Different content types can benefit from different IW sizes, and our results suggest that content-dependent customizations (e.g., for interactive video streaming) exist. Still, they cannot be purely detected by MIME type since they more likely depend on the delivery strategy selected at the CDN.*

3.1.5 Worldwide Perspective on CDN IWs

To investigate if CDNs tailor IWs to networks, we probe the same URL from multiple vantage points. To do so, we utilize the public VPNs listed at VPNGate [NS14]. As the service lists thousands of VPNs, we concentrate on a small subset of 14 VPNs all located in different countries and ASes. For these VPNs, we test samples of URLs (5 per IW/CDN combination) for which we have already established an IW from our previous campus network-based measurements, thus enabling to compare if other networks are subject to different IW configurations.

#VPN	ASN	AS Name	Country	Link Type
1	AS1221	Telstra	Australia	Consumer
2	AS3303	Swisscom	Switzerland	Consumer
3	AS3326	Datagroup	Ukraine	?
4	AS4766	Korea Telecom	South Korea	?
5	AS7552	Viettel	Vietnam	Consumer
6	AS7922	Comcast	USA	Consumer
7	AS9198	Kaztelecom	Kazakhstan	Consumer
8	AS12389	Rostelecom	Russia	Consumer
9	AS16276	OVH	France	Datacenter
10	AS17534	NSK	Japan	?
11	AS24560	Airtel	India	Consumer
12	AS24620	Riga Tech. Univ.	Latvia	University
13	AS28548	Cablevisión	Mexico	Consumer
14	AS28885	OmanTel	Oman	Consumer

Table 3.5 Classification of VPNs used to estimate CDN IW configurations.

VPNs	Akamai		Azure	Cachefly	Cloudf.	Edgecast	Fastly		Highw.	Level 3
	16	32	30	105 kB	25	30	100	10	64 kB	32
1,9	1	1	1	1	1	1	1	1	1	1
2-6,8	16	32	30	105 kB	25	30	61-62	10	64 kB	32
7	16	32	20-30	105 kB	25	20-30	61-63	10	20-60 kB	32
10	16	32	30	105 kB	25	30	1-100	10	83 kB	32
11	16	32	30	105 kB*	25	15-30	2-61	10	4-49 kB	32
12	16	32	30	105 kB	25	30	87-99	10	64 kB	32
13	16	32	6-30	105 kB	25	2-30	6-73	10	64 kB	32
14	16	32	30	105 kB	25	30	61-75	10	58 kB	32

Table 3.6 IW configurations observed at the VPNs. The top row shows the CDNs with their IWs as discovered within our campus network. Each field marks the IW we discovered through the VPN or a range if we saw consistent losses. Results marked with (*) experienced packet loss but no tail loss.

Table 3.5 gives an overview of the VPN locations (as reported by VPNGate), networks as well as a manual classification of their link's nature. We classified the link type by inspecting *i)* the AS and *ii)* the rDNS name of the VPN host and check if it includes keywords such as cable, (A)DSL, dynamic, or others. Most of our VPNs are located in residential access networks, except for one in a datacenter (#9), one in a university network (#12) and three links (#3, #4, #10) that we could not classify due to missing hints.

Results

Table 3.6 summarizes our IW estimations through these VPNs. We were able to build classes of VPNs that perform similarly, already indicating that many of our VPNs show similar performance and we see a similar IW configuration. The first

class for VPNs #1 and #9 shows the most significant divergence from our campus network. Here we measured an IW of only one segment for all CDNs contacted via both the consumer (#1) and the datacenter (#9) link. Especially for a datacenter link, this seems too low and does not fit the rest of our data. When more closely inspecting both VPNs, we found that both VPNs seem to heavily rate-limit the packet-rate. Even when performing a regular download of the URLs, we are unable to get more than two segments in a roundtrip at any time. Thus, we believe that the IW estimation here is unable to determine the actual IW due to the rate-limiting, which highlights the challenges when using vantage points that are out of direct control.

The second, largest class, of VPNs, paints a similar picture to that of our local observations. We observe for all but one CDN provider the same IWs as seen from our university network. The only difference being Fastly, for which we have measured IWs between 61-62. Here, we consistently measured IWs in that range, indicating that our measurements are subject to loss. We take this as an indication that likely, 62 is not the actual IW that should have been delivered but rather a larger IW was subject to substantial tail loss, especially since all other IWs are configured similarly to our local observations.

This impression continues when observing the remaining VPNs, there the IWs for Fastly also reach up to 100 segments (VPNs #10 and #12) but with consistent losses between multiple measurements. For many, we observe IWs in the range of 60 to 70 segments. We take this as an indication of a service-specific configuration rather than a network-dependent one.

But we also find patterns of network-dependent configuration, e.g., for the Highwinds CDN that we measured with an IW of 64 kB locally. For VPNs #10 and #14, we consistently observe different IWs. For VPN #10, we observe a larger IW of 83 kB and for #14 only 58 kB. Also for Highwinds, we can observe losses at VPN #7 and #11.

Especially, VPN #11 observes the highest losses throughout our measurements. Here, also Cachefly with the second-largest IW (equaling to 72 full-sized segments) that we observed shows losses (which does not show any losses at other VPN).

Limitations

Even though, we likely see signs of network-driven tailoring behavior, we could be hitting legacy systems that are just differently configured. Furthermore, since we use public VPNs, other requests could affect our IW-estimations that were done over this VPN. However, this would actually strengthen our observations that CDNs in-fact do tailor.

Takeaway. *Overall, we observe that many CDNs use the same IWs regardless of the network and are successful in delivering it without losses. Interestingly, we find that Level 3 and Akamai both deliver content without loss using IW32, while others like Edgecast and Azure experience loss over the same links despite using a smaller IW of 30.*

Motivated by these observed losses, we want to investigate the burstiness of IWs. To this end, a recent proposal [All15] recommends using TCP pacing (see Section 2.3.3.4 for a detailed rationale) to evenly space out packet delivery over the RTT when exceeding an IW of 10 segments to be less aggressive towards queues. Visweswaraiah and Heidemann [VH97] have also proposed to use pacing after idle slow start restarts. Since Linux Kernel 3.11 (released in September 2013), it offers pacing support via a selectable packet scheduler in the traffic control (TC)-subsystem, starting with Linux Kernel 4.13 (released in September 2017) also directly from within the TCP stack. Thus, we continue to inspect the temporal characteristics of the packets transmitted in the IW to investigate the use of pacing at CDNs.

3.1.6 Burstiness of the CDN IWs

To investigate the use of TCP pacing by CDNs, we again focus on our university network as we require fine-grained packet arrival times, which are not preserved through the VPNs.

Traffic Shape of the Linux Pacer

Linux implements pacing (see Section 2.3.3.4 for a general introduction to pacing) in TCP either via TC as a queuing discipline or directly from within the TCP stack. For TC, the default queuing discipline must be exchanged to use the TC flow queuing (FQ) discipline. TCP interacts with FQ by setting an appropriate pacing rate on the socket that is used by FQ to spread out packets. TCP re-calculates the pacing rate with every incoming ACK and after the initial handshake as $\frac{MSS \cdot cwnd}{sRTT} \cdot RATIO$ with MSS, the cwnd in segments, the smoothed RTT (sRTT), and a RATIO defaulting either to 200% in slow start or 120% during congestion avoidance.

The ratio is used to accelerate the pacer in different connection phases, and `sysctl` parameters allow adjusting it, for both, slow start and congestion avoidance. The pacing rate defines the amount of data that the stack can send, and the pacer uses this rate to calculate the departure of the next packet such that the rate is enforced on the connection. In addition to enforcing a bandwidth, the pacer also allows for a certain burstiness of the traffic.

This burstiness is enabled by combining the previously discussed rate with a token bucket scheme. The scheme assigns a *credit* to each flow that initializes using an *initial quantum*. Whenever the algorithm dequeues a packet, it checks if the credit is zero or below and, in case it is, enqueues the packet again while increasing the credit by a *refill quantum*. Thus, new flows, i.e., at the start of the connection, may directly burst the initial quantum and only then are subject to the rate limiter and will continue sending bursts of packets governed by the refill quantum (assuming enough data is available for dequeuing).

Figure 3.13 shows the resulting traffic pattern with the pacer's default values for the initial quantum, i.e., 10x MSS, and the refill quantum, i.e., 2x MSS, for a server serving data at the start of the connection with IW10 and IW20. We used *NetEm*

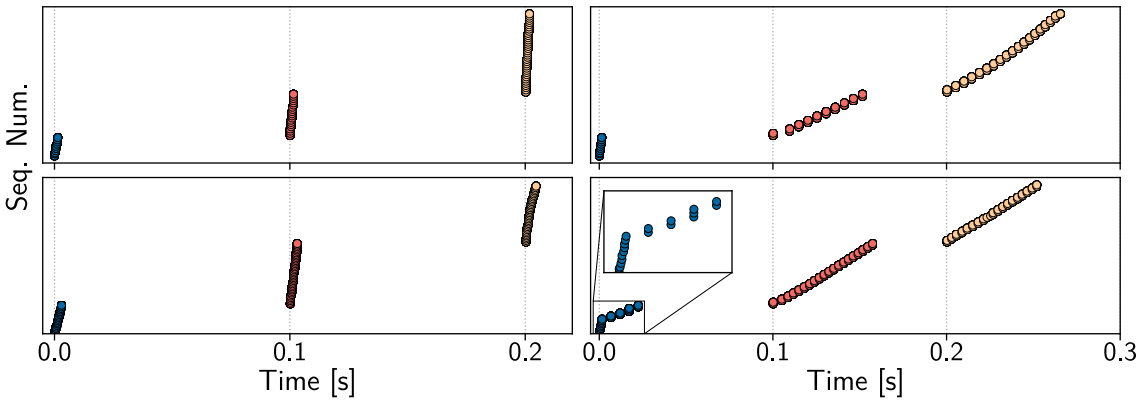


Figure 3.13 Linux default traffic pattern on the left, pacing using FQ on the right. Each marker denotes one packet. Packets arriving in the same roundtrip have the same color. Top row uses IW10, bottom IW20.

to add a delay of 100 ms at the egress of the client to simulate an RTT of roughly 100 ms in our local network and then measured the packet arrival. As we found that the packet coalescing of the network interface card (NIC), which reduces the interrupt-rate, causes imprecise software timestamps of the arriving packets, we instruct our NIC to perform hardware timestamping at packet arrival.

To illustrate the difference when using pacing, the left side of the figure shows the typical, bursty behavior without packet pacing. As we can see on the right side, the default pacer parameters are chosen in a way to allow bursts of 10 packets, thus allowing an IW of 10 segments (i.e., the IETF-recommended value) to seamlessly pass through the pacer, while bytes beyond IW10 are subject to pacing as visible in the lower right plot that uses IW20.

These trains of two packets (refill quantum) correspond to TCP's self-clocking [Jac88] behavior when having delayed ACKs that generate an ACK for every second segment thus causing the release of two packets with every incoming ACK.

Whether or not pacing like this is a good strategy is outside of the scope of this dissertation, though, we remark that additional research in this area is desirable to better understand the impact of the initial burst and the microbursts, especially when regarding features like frame-aggregation in wireless and mobile networks. We next empirically probe CDNs for this pattern to detect pacing.

Measuring the Packet-Pacing

To measure if the CDNs utilize pacing, we take a look at the packet arrival-times when executing an IW scan. To do so, we record packet traces (with tcpdump) but instruct our IW-prober to delay the ACK following the SYN/ACK by roughly 50 ms to emulate a larger RTT to the measured CDN. We again use hardware timestamps for precise timekeeping.

Similar to Figure 3.13, Figure 3.14 depicts the IW as the received packets (dots) after connection start for a selected subset of CDNs and URLs. Their arrival time is

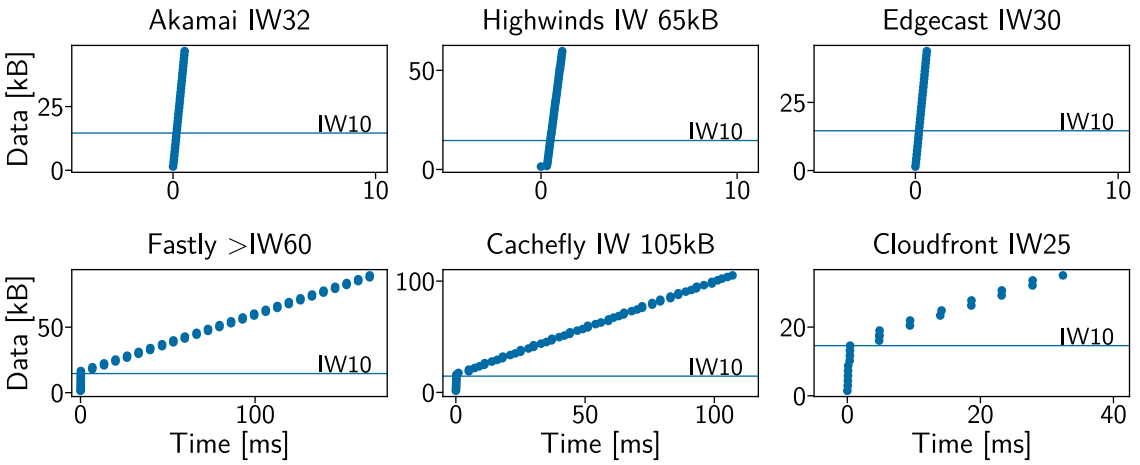


Figure 3.14 IW burstiness for a subset of the observed CDNs and URLs (each with an RTT of 60 ms-70 ms). The arrival time of full-sized 1500 B packets (dots) in the entire IW is shown on the x-axis, the IW size (in kB) on the y-axis (e.g., IW10 = 15 kB). Note different axis scalings due to different IW sizes. Some CDNs seem to utilize packet pacing while others do not.

depicted on the x-axis and the IW size in kB on the y-axis. Please note the different x- and y-axis scaling due to the different IW sizes. We can visually observe two different patterns. The first, here presented by Akamai, Highwinds, and Edgecast (top row), shows close to no temporal distribution of packets. The second, presented by Cloudfront, Fastly, and Cachefly (bottom row), shows a stream of packets arriving virtually at the same time followed by a temporally skewed train of other packets. The latter follows the expected output of Linux's packet pacer as described before (Figure 3.13). This behavior is best visible in the example of Cloudfront, where a burst of ten segments is almost perfectly followed by delayed trains of two packets. Thus, we can see that some CDNs are likely utilizing pacing during slow start for IWs larger than IW10 as recommended in [All15].

IW-Skew by Pacing Rate

When looking at the two largest IWs that we observed by Cachefly and Fastly[13], we can see that both pace their IWs, however, we observe that Cachefly is more aggressive in doing so as they spread their IW over roughly 1.5x the RTT while Fastly does it over roughly 2.5x the RTT. Compared to smaller IWs like observed at Cloudfront that pace the IW over roughly 0.5x RTT (i.e., very close to the Linux pacer's default), it might not be obvious why spreading the data transmission over more than an RTT would make sense. We speculate it could be favorable in situations where the initial RTT sample from the three-way handshake is a poor estimate for the RTT or when there is congestion on the reverse path. When the initial RTT is lower than the RTT that the CDN expects during the transmission, e.g., when there is other traffic filling queues or one must compete for airtime in wireless settings, it might take more time for ACKs to arrive than the initial RTT sample predicted.

[13]Please note that while measuring pacing, we experienced heavy tail-loss with Fastly leading to the reduced bytes.

Similarly, when there is loss on the reverse path, ACKs might not arrive rendering the connection idle, and thus, in both cases prolonging the IW transmission duration *could* be favorable.

This kind of pacing challenges the traditional notion of IWs. Typically, the IW is thought of as the number of bytes sent in the first RTT without requiring an acknowledgment. Since traditional TCP is bursty, pacing skews this notion as the data must not necessarily arrive within the same RTT as we already observe for some CDNs. Thus, e.g., when an IW of 100 segments is paced over two RTTs, the bytes arriving at the receiver in the first RTT are effectively half of what is received with a bursty IW100. Therefore, depending on the pacing rate, a paced IW100 might better compare to a bursty IW50 or even less. Given this observation, discussing IWs and merely looking at the number of segments is insufficient to reason about its appropriateness. One should regard IWs concerning time and rate, e.g., an IW of 100 segments paced over two RTTs with an RTT of 30 ms corresponds to a rate of ∼20 Mbit/s which seems reasonable when looking at the capacity of current user access speeds. While this does not capture the rate on a sub-RTT level, e.g., when TCP uses no pacing, a data rate better captures the demand of a new connection on the network than a fixed number of segments.

Takeaway. *We find it is likely that some CDNs use pacing. The two largest IWs show distinct pacing patterns. Past research suggests that pacing can help to bootstrap new or idle connections. However, there is currently only a limited understanding of the impact of pacing on networks and of its benefits and drawbacks, especially as current pacers deviate from perfectly paced packet streams found in simulation literature. Additionally, pacing challenges the way one should regard IW values; we find that a data rate better captures the demand on a network.*

3.1.7 IW Performance when Competing for Traffic

While we initially investigated the theoretical advantages and disadvantages of using larger IWs (Section 3.1.1.1), these measurements were idealized. Further, our previous CDN measurements have shown that some CDNs utilize pacing to distribute the initial load on the network over a more extended period. We take these observations and investigate how the parameters affect performance in a more realistic setting, i.e., when having to compete for bandwidth. Even though we are not the first to investigate the performance impact of larger IWs, other studies often use simulations and do not rely on the actual code and configurations that we observed in the wild. Still, there are many possibilities to investigate this, e.g., how do the parameters affect a congested peering link or how do they affect performance in a data center. We opt to investigate the effects from a user's perspective. Even though it is known that there are instances of persistent inter-domain congestion [DCG+18], it is still widely believed that much congestion happens at the network edges and more specifically at the end-user's access link [BCL09], i.e., the last mile. We use this setting and investigate the performance on an emulated link where a new, comparably short flow must compete against an elephant flow, i.e., a bulk transfer, reflecting a typical situation at home with a shared access link, i.e., a bulk download competes against

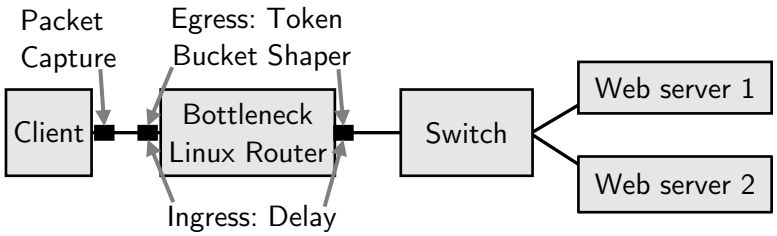


Figure 3.15 Testbed topology with the client requesting traffic from the two Web servers. Bottleneck characteristics are configured on the dedicated bottleneck machine. Packet traces are captured on the client machine.

Web traffic. We start our evaluation by describing our testbed before continuing with discussing our results.

3.1.7.1 Testbed and Parameterization

We extend the testbed used in Section 3.1.1.1 by implementing a simple dumbbell topology, illustrated in Figure 3.15. It allows us to reflect our end-user setting and enables us to investigate CC adequately [FK03]. For our measurements, the client machine to the left of the figure requests a high volume elephant flow from Web server 2. After it has reached the bottleneck's capacity, the client requests a second, small volume flow from Web server 1.

We again vary between four different bandwidth and three different delay configurations motivated by Akamai's report [Aka16]. Even though CDNs, in general, strive for low RTTs of a couple of 10th of ms, wireless links and areas with suboptimal CDN coverage may still face higher RTTs. Additionally, we size the bottleneck's queue following the bandwidth delay product (BDP) rule of thumb (see Section 2.3.4 for an overview of buffer sizing). Since, we know of no studies that investigate how router buffers are sized in the Internet, especially at the edge, we use 0.5x BDP, the BDP itself, and 1.5x the BDP.

We configure the bandwidth and queue size using a token bucket filter with a burst size of a single frame at our Linux-based bottleneck machine while using a traditional drop-tail FIFO queue. Even though Internet access links are often asymmetrical, we disregard this fact as we are not interested in investigating reverse-path congestion and use the same bandwidth in both directions.

To add delay to our testbed, we modify our bottleneck's ingress packet processing. There, we artificially redirect traffic to an intermediate queue disc enabling us to use NetEm to add delay before we release the packet for forwarding to the actual egress queue. While care needs to be taken to size the NetEm queue to not cause artificial packet loss this approach has the advantage that the end-host stacks are not involved in the delay which is known to harshly interfere with CC when Linux detects queuing pressure (TCP small queues). To have a symmetric delay, we add half of the configured delay to each ingress of the bottleneck. We do not configure any artificial jitter using NetEm as this causes packet reordering, the delay and jitter are thus only caused by the egress queue.

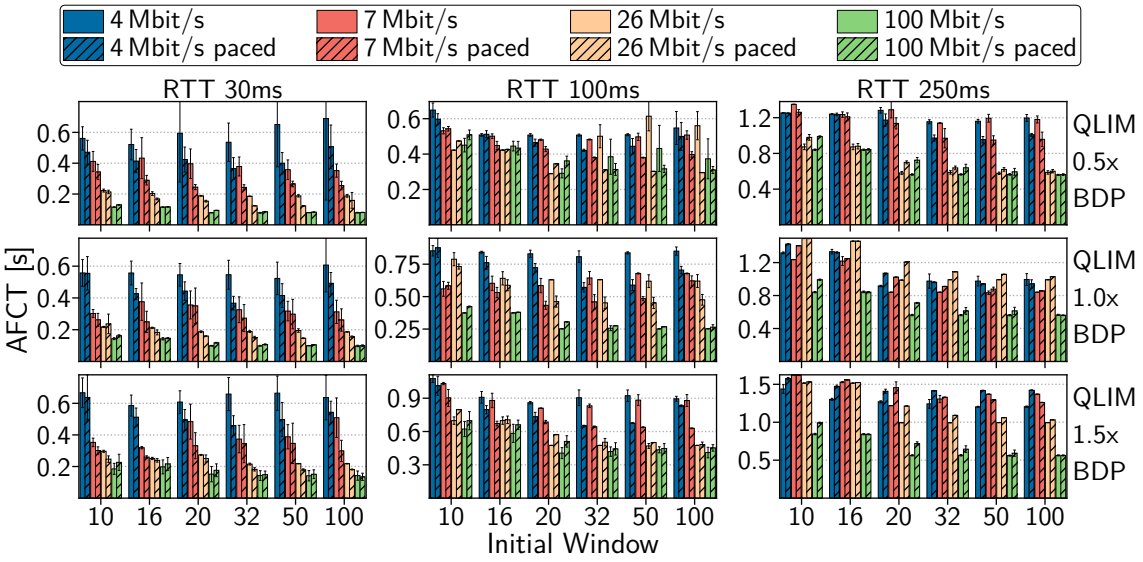


Figure 3.16 AFCT for a 73 kB CUBIC flow (solid) and a paced CUBIC flow (hatched) when competing against a CUBIC elephant flow for different RTTs (columns) and different queue sizes (rows) subject to bottleneck bandwidths (colors) when using different IWs (x-axis), errors bars denote 95% confidence intervals over 30 measurements.

All our machines are connected via Gigabit Ethernet and use a Linux 4.13 kernel. Further, we make sure that the initial rwnds are large enough, not limiting the IW (see Section 3.1.4.1). Additionally, we clear all TCP metrics after each measurement and make sure that send and receive buffers are sized such that the machines can fully utilize their Gigabit Ethernet connection.

We capture traffic at the client using tcpdump to compute the AFCT of the short flow.

3.1.7.2 Increasing CUBIC IWs and Applying Pacing

Our first study investigates the advantages when utilizing increased IWs as observed in our CDN measurements and the additional implications that come with pacing. To this end, we investigate how CUBIC (see Section 2.3.3.2 for a description), the Linux default congestion control algorithm, performs subject to larger IWs and pacing. Slightly different to our initial investigations, we use 73 kB of data for our short flow (to have slightly more than 50 frames) and this time our sole focus is on the FCT. After the elephant flow has started, the client requests the short flow, which then competes for bandwidth and for which we measure the AFCT. We repeat each configuration 30 times to be able to investigate the results statistically. Figure 3.16 visualizes our study, each row showing the same measurements subject to different bottleneck queue sizes, each column for different RTT configurations.

Bursty TCP

We first focus on the non-paced performance (i.e., all non-hatched bars), starting with the 30 ms column reflecting a well connected CDN. Our first observation is best visible at 4 Mbit/s. Here we observe that for this low-end bottleneck speed, increasing the IW results in decreased performance. On average, the FCT increases with increasing the IW, especially when the queue size is small, yet regardless of the queue size (rows), the stability decreases as indicated by the increasing confidence intervals. Thus, while some measurements show increased performance, some show extremely worse performance indicating the unsuitability of large IW for these network configurations.

Similarly, no real gains are observable for 7 Mbit/s bottlenecks, only starting with 26 Mbit/s, especially for larger queue sizes. Things do slightly change when looking at higher RTTs. At 100 ms slight IW increases to 16 or 20 segments still yield performance increases even for low bandwidths. Nonetheless, we observe that IWs such as 50 or 100 cause problems at 26 Mbit/s and 100 Mbit/s indicated by increased variance, especially visible for short queue sizes. With an RTT of 250 ms, our measurements indicate that for small queues, again sufficient bandwidth is required to utilize larger IWs while not hurting performance. When increasing the queue size, performance generally decreases as in all other settings, however, we see much clearer that the long flow hogs the queues leading to reduced performance even when having larger bandwidths.

Paced TCP

We next shift our focus to the performance of the paced CUBIC short flow (i.e., all hatched bars). Looking at 30 ms RTT, we find that pacing, in comparison to its bursty counterpart, enables TCP to utilize an increased IW already for 4 Mbit/s bottlenecks. First, the FCT, on average is generally lower than the bursty variant, second, at times where the IW starts to worsen the performance in the unpaced setting, it still improves the performance up to an IW of 32 segments. Until then, the FCT even compares to the unpaced variant with 7 Mbit/s. After that, the performance again starts to worsen, notably with an IW of 100 segments (recall we are only transmitting slightly more than 50 frames), the performance notably worsens in comparison to the IW50 case. We believe this is because TCP uses the IW of 100 segments to calculate the packet departure times even though much less is transmitted leading to a smaller inter-packet gap and overall shorter time frame which seem to put too much pressure on our bottleneck. This observation carries over to larger bandwidths as well, however, not as drastically visible. For 100 Mbit/s, we even see that the performance slightly worsens, this is not without surprise, the last packet will depart roughly half an RTT later in the paced variant. This especially makes a difference when there is less congestion. Pacing's advantages slightly diminish over larger buffer sizes, yet, in most cases, it is still beneficial.

For larger RTTs of 100 ms and 250 ms, pacing does not always yield improved performance. Especially, when having larger queue sizes that absorb bursty traffic (see 1.0x and 1.5x BDP at 250 ms), it seems that the competing elephant flow leaves

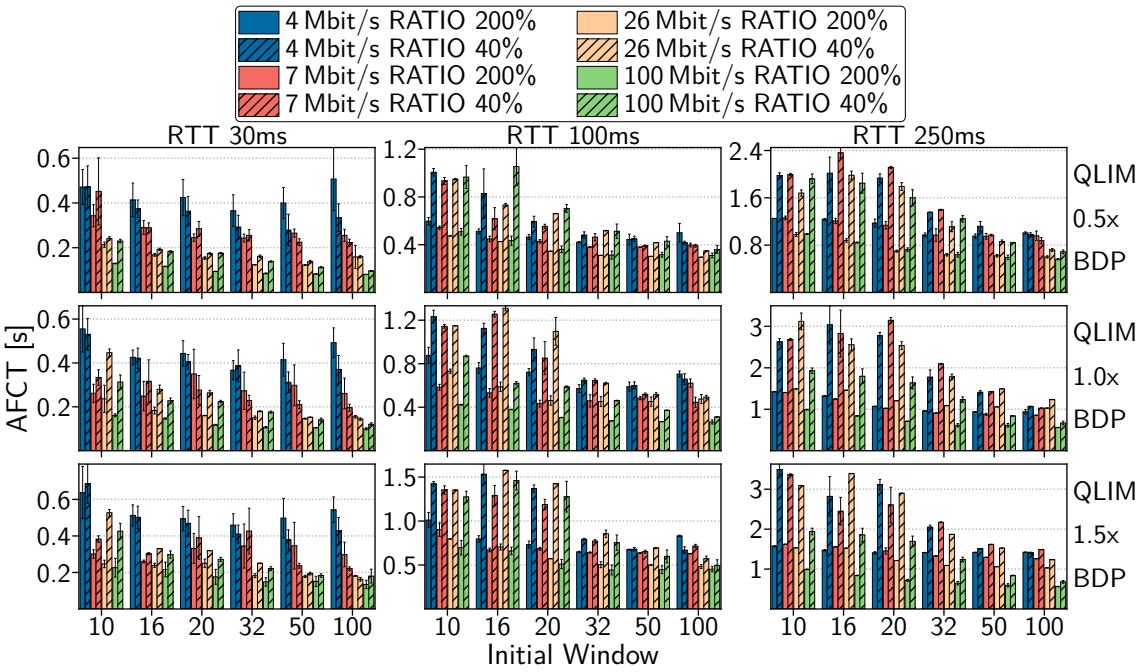


Figure 3.17 AFCT for a 73 kB paced CUBIC flow with a slow start RATIO of 200% (solid) and a paced CUBIC flow with a slow start RATIO of 40% (hatched) when competing against a CUBIC elephant flow for different RTTs (columns) and different queue sizes (rows) subject to bottleneck bandwidths (colors) when using different IWs (x-axis), errors bars denote 95% confidence intervals over 30 measurements.

enough buffer space during collision avoidance that the short flow can pass through without requiring pacing. This observation is in line with simulation results from related works that show pacing's benefits, especially when having small buffers.

Takeaway. *Our investigations indicate that pacing enables TCP to utilize larger IWs when competing against an elephant flow. These advantages are especially apparent with small buffers and short RTTs while they diminish for larger bandwidths, because pacing, by its principle, worsens the flow completion time in uncongested settings.*

3.1.7.3 Pacing Aggressiveness in Slow Start

In our CDN observations about pacing, we already saw that some CDNs spread their IW over different fractions of the RTT. By default, as used in the previous section, Linux's pacer paces over 0.5 of the RTT by setting the RATIO to 200%. Since we already observed how larger IWs that are not utilized by application data affect the pacing performance in the previous section, we want to more thoroughly investigate how this *aggressiveness* of the pacer affects the performance. To this end, Figure 3.17 compares our previous paced variant with a RATIO of 200% against one where we set the RATIO to 40%, i.e., such that the pacer spreads the IW over 2.5x the RTT, which was the most extended spread observed in our previous measurements. For the 200% RATIO, we reuse the data from before (now non-hatched part of the plot).

Looking at the first column (30 ms RTT), we observe that spreading the IW over a prolonged time can yield a slight performance increase for low-bandwidth settings

even though using large IWs. We believe that the large spread leads to a later exit of slow start even though using a large IW, ultimately shortening the FCT for this low bandwidth setting. When we look at larger bandwidths, we can see that the decreased pacing rate hurts performance, especially for small IWs. Interestingly, the reduced pacing rate is beneficial when the IW is larger than the application data (IW 100 in this case). However, we think it is not the correct measure to reduce the aggressiveness of too large IWs when there is few application data. Preferably, the pacer should be informed about application data limitations and use the available amount of data as the IW if it is smaller than a preconfigured IW.

Focusing on the larger RTTs, we can observe that in nearly all tested settings, the prolonged transmissions cause significant increases in the FCT. Only for IWs of 50 or 100 segments, the prolonged transmission time seems to nearly amortize. Thus, the ideal configuration of the pacer seems to dependent profoundly on the concrete application and scenario.

Takeaway. *We tested a very extreme acceleration change of 200% down to 40%. Although a smaller RATIO, hence longer duration, seems to be beneficial in some situations, it is disadvantageous in most of the investigated settings. Nevertheless, accelerating or decelerating the pacer in different settings or throughout the connection seems to be an area worthwhile to further explore, a property inherent to the BBR CC.*

3.1.7.4 Increased IWs with BBR Congestion Control

For our final investigation of IW performance, we switch the CC algorithm. Now, we utilize the recent BBR algorithm (see Section 2.3.3.3 for an overview of BBR) that is known to be used by Cloudflare and Google. Its design builds around the idea of pacing, however, compared to our paced CUBIC that we tested before, we cannot adjust the pacing rate in slow start which is called startup in BBR, see [Goo19c] for an in-depth analysis of the BBR startup parameterization. We remark that compared to paced CUBIC, BBR chooses the pacing rate and calculates the congestion window, such that it doubles with every RTT even though not all packets have arrived yet (due to the pacing), effectively leading to a slightly different packet release mode than paced CUBIC.

Despite these small differences, BBR is still very similar to paced CUBIC (in slow start), which is why we compare our default paced CUBIC variant (200% RATIO) to BBR in Figure 3.18. BBR overall compares well to our paced CUBIC variant, yet, there are subtle differences. Looking at an RTT of 30 ms, we see that for many parameterizations BBR's average in our experiments is below that of paced CUBIC. However, as indicated by the mostly overlapping confidence intervals, this is not necessarily statistically significant. IW100 stands out; it seems that for lower bandwidths, BBR scales better to the actual application demand (remember we are only sending slightly more than 50 frames). For an RTT of 100 ms, paced CUBIC and BBR perform very similar. Looking at 250 ms, we find many instances where BBR's FCT is higher, especially when looking at the larger buffer sizes. It seems that

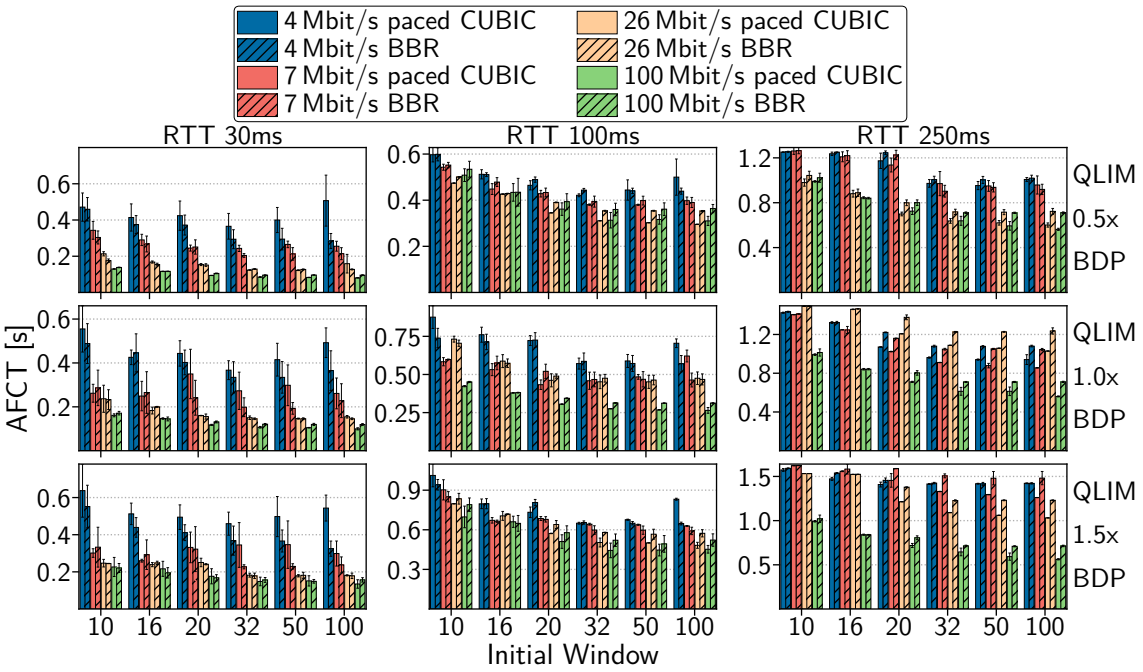


Figure 3.18 AFCT for a 73 kB paced CUBIC flow with a slow start RATIO of 200% (solid) and a BBR flow (hatched) when competing against a CUBIC elephant flow for different RTTs (columns) and different queue sizes (rows) subject to bottleneck bandwidths (colors) when using different IWs (x-axis), errors bars denote 95% confidence intervals over 30 measurements.

the delay brought in by the CUBIC elephant flow seems to inflate the BBR-observed RTT further, causing it to reduce its sending rate.

Takeaway. *BBR's startup compares to our paced CUBIC variant. BBR seems to have slight advantages at lower bandwidths and especially if the cwnd exceeds the actual application demand. For large RTTs with larger buffers, it seems that our paced CUBIC variant has advantages when the IW increases.*

3.1.8 Summary and Discussion

This contribution's goal is a better understanding of the current configuration and evolution of TCP's IW and how Internet giants push and challenge conventional wisdom. The IW is a long-debated performance parameter. Its size is in principle network- and application-dependent, where too small IWs can add unnecessary latency and too large IWs can cause congestion and thus loss. Nevertheless, the IW is regarded as a *static* parameter that fits all networks and applications. Its IETF-recommended size has only changed infrequently in its history.

To this end, we utilize longitudinal IP address-based scans of IPv4 and the Alexa list to study the adoption of IETF-recommended IW values and find that IW10, as the current recommended value, gains more and more track. We have shown that we can reduce the impact of our world-wide scans drastically when only scanning a random 1% sample, which allowed us to monitor the IW evolution over more than two years. Further, we modify our scanner architecture to measure how Internet

giants utilize IWs. To this end, we take a look at CDNs as major CPs from our University network and globally distributed vantage points. We find that CDNs are well ahead of current IETF-standardized practices by using *custom* IW configurations. In our measurement study, we observe IW configurations that are up to ten times higher than the most recent experimental standard. Our results suggest that CDNs do customize IWs for different services or customers, yet while advantageous for some content types, the content type does not enforce the IW. On a larger scale, we survey if CDNs adjust IWs depending on the end-user's network. We find some CDNs for which we can show that IWs vary depending on the network, but not for all. Driven by losses in our measurements, we analyze the burstiness of the IW delivery and find that some CDNs utilize pacing to space out packets over time. We find that the largest IWs in our study utilize this feature, which especially challenges the notion of IWs as they must not arrive within the first RTT, making it tough to compare IWs just by the number of segments. We find it reasonable to couple an IW to a data rate, i.e., over which time is this IW transmitted, to better understand the IW's demand on the network.

Since our measurements do not show if pacing actually enables these large IWs, we carry our real-world observations to a controlled lab evaluation and investigate their impact on FCTs when competing against a long-lived flow. Our testbed study indicates that pacing is a curcial component to enable larger IWs, yet, it still shows that blindly increasing the IW without regarding the actual network and application results in decreased performance.

While our study focuses on TCP, QUIC borrows TCP's CC and startup phase, including IWs highlighting its future relevance (also in light TCP-BPF [Bra17]). We have thereby shown that standardization and academia have disconnected from current global trends and especially Internet giants such as CDN depart from current knowledge and IETF-recommendations. Still, while departing from standardized practice, it seems that many Internet giants are well-aware of the repercussions that come with tuning TCP parameters. Our next contribution highlights how Internet giants push forward QUIC as a new transport for the Internet that overcomes the ossification of TCP, enabling end-to-end modifications of the transport and not only sender-side changes.

3.2 Deploying a New Internet Transport – QUIC

Recent years have fostered the understanding that TCP as the de-facto default Internet transport protocol has become a technological bottleneck that is hard to evolve (see Section 2.1). This understanding is rooted in the fact that optimizing throughput is no longer a key concern in the Internet, but optimizing latency and providing encryption at the *transport* has become a major concern. The focus on latency results from shifted demands (e.g., by interactive Web applications) and is currently proposed to be addressed in part by TCP extensions at the protocol level, e.g., TCP Fast Open [RCC+11] or Multipath TCP [RPB+12]. While optimizing latency, there is an additional demand to also provide an encrypted transport, typically realized by TLS on top of TCP. Since this additional encryption adds additional latency, further optimizations address this latency inflation, e.g., 0-RTT in the TLS 1.3 standard [RFC8446]. While these approaches present clear advantages on paper, middleboxes and legacy systems currently challenge their deployment.

Google's QUIC[14] protocol [LRW+17] aims to address these shortcomings in a new way (see Section 2.3.2 for an in-depth description of QUIC). Like TCP, it provides a connection-oriented, reliable, and in-order byte stream. Unlike TCP, it enables stream multiplexing over a single connection while optimizing for latency. By fully encrypting already at the transport layer, QUIC provides security and excludes (interfering) middlebox optimizations; thereby paving the way for a rapidly evolving transport layer. By implementing QUIC in user space on top of the User Datagram Protocol (UDP), its ability to rapidly update and customize a transport per application has yet unknown consequences and motives measurements. It was first introduced to Chromium in 2012 and has undergone rapid development and high update-rate since then — as we will partly show in our measurements. Since 2016, the IETF QUIC working group [IET17b] is working on its standardization as IETF QUIC (iQUIC). Google widely enabled Google QUIC (gQUIC) for *all* of their services' users in January 2017 [Swe16, LRW+17], motivating our study, capturing its first months of general deployment. Yet, in contrast to TCP and TLS, there is only minimal tooling support to analyze QUIC and the academic understanding is currently limited to protocol security [FG14, LJB+15, JSS15] and performance [CDM15, CMT+17, LRW+17, KJC+17].

While QUIC seems to be a promising candidate to overcome TCP's deployment challenges, it does not come without leaving a bitter aftertaste. Network operators, especially mobile operators, fear a vast deployment of QUIC [IET17a]. For decades network operation have been relying on the availability of transport headers to measure the quality of their networks. To this end, TCP sequence numbers and ACKs are used to derive RTTs, detect loss and reordering, all key performance indicators (KPIs) for network operators. With QUIC, such information is hidden from a passive observer through heavy encryption. As we will show, Internet giants already challenge the operation of networks today. Especially in light of HTTP/3, the next generation of the application transport for the Web, which specifies QUIC

[14] Initially an acronym for Quick UDP Internet Connections.

as the underlying transport, network operators are challenged when more and more traffic moves over to QUIC.

In this contribution, we expand on the understanding of QUIC by providing the first large-scale analysis of the current QUIC *deployments* and its *traffic share*, thereby observing how Internet giants transform the transport layer. Subsequently, we analyze whether QUIC can actually deliver its promises of a faster transport for the Web by performing extensive testbed evaluations and user studies. But first, to assess the QUIC deployment, we have been performing regular probes of the entire IPv4 space for gQUIC support since August 2016. We further complement our IP-based scans with an in-depth view of gQUIC in October 2017 where we additionally probe the complete set of .com/.net/.org domains as well as the Alexa Top 1M list, i.e., around 46% of the domain name space [Ver17]. In September 2018, we performed an in-depth study on the fingerprinting potential of gQUIC servers, and since August 2018, we further started monitoring iQUIC to grasp an understanding of how standardization affects QUIC's evolution.

To assess the traffic share that these deployments generate, we analyze traffic traces from three vantage points: *i)* 9 months of traffic in 2017 on a transit link to an ISP (MAWI dataset [MAW18]), *ii)* several days in 2017 and 2018 at a European Tier-1 ISP, representing edge (DSL + cellular) and backbone traffic, and *iii)* one day in August 2017 at a large European IXP. We complement this analysis and follow up on traffic shares of the ISP by investigating 13 additional days that, in total, are spread over one full year to understand how QUIC traffic shares evolve and thereby provide a neutral source of information for standardization, Internet giants and network operators.

To analyze whether QUIC can keep its promises of accelerating the Web delivery, we source from our observations from our previous contributions. We found that Internet giants are well ahead of IETF-recommended practices, e.g., utilizing large IWs and pacing. In fact, these observations have also found its way into the de-facto reference implementation of QUIC, i.e., Google's QUIC implementation in Chrome. Further, previous works have neglected these observations and have thus compared a highly tuned QUIC stack against a TCP stack that is configured for stability and not for performance. We recognize these shortcomings and compare both protocols on an eye-level, i.e., we parameterize both protocols in a similar fashion. First, we regard their performance in an emulated testbed using purely technical metrics, and subsequently we perform two extensive user studies to shine a light on the QoE for actual end-users.

With this section, we contribute to the overall understanding of the evolution on the transport layer as follows:

- We analyze the development and deployment of gQUIC and iQUIC in the IPv4 Internet.
- We present the first comprehensive view on QUIC deployment and traffic outside of Google's network from three different vantage points.

- We build and publish tools to enumerate QUIC hosts and to massively grab and decode QUIC protocol parameters, which we further use to study the potential to fingerprint server software, provider, or service type.
- We publish all our active measurement data and future scans via [Rüt19].
- We provide the first study that performs an eye-level comparison of TCP+TLS+HTTP/2 and QUIC. Our study highlights that QUIC can indeed outperform TCP in a variety of settings but so does a tuned TCP stack.
- Our study is unique in that it is the first to assess the QoE of actual users in two large-scale user studies, and in combining it with a pure technical evaluation.
- Tuning TCP closes the gap to QUIC and shows that TCP is still very competitive to QUIC.
- We find that users actually do perceive QUIC as the faster protocol in a direct side-by-side comparison even against tuned TCP stacks.
- However, in isolation, users generally do not prefer one protocol over the other if the network is sufficiently fast.
- In slow and lossy networks, QUIC's advanced protocol design seems to enable a more satisfying loading process for our study participants.

Structure. Before diving into the scans, we first present an overview of related works and then provide an overview of the different datasets and measurement campaigns that we gathered and performed in Section 3.2.2. Subsequently, as a foundation for QUIC host enumeration, Section 3.2.3 introduces the gQUIC handshake. Section 3.2.4 presents our view on QUIC in IPv4, in three large top-level domains (TLDs) and the potential for fingerprinting servers as well as the tools that drive our measurements. Section 3.2.5 shows how QUIC reshapes traffic in campus and ISP/IXP networks. After analyzing the penetration of QUIC in the Internet, Section 3.2.6 analyzed if this evolution is worth the effort from a performance point-of-view. Finally, Section 3.2.7 concludes this contribution.

3.2.1 Related Work

Even though Google already introduced QUIC in 2012, it has not been the subject of many studies. There is only limited work on QUIC's evolution; for the most considerable part, academia has analyzed QUIC for its security measures and performance. We now give an overview of how QUIC has been studied in these three categories.

QUIC Performance. QUIC performance is subject to a body of studies [BG16, CDM15, CMT+17, KJC+17, MKM16, YXY17, NdOA+18, SSW+19b], most compare QUIC against some combination of TCP+TLS+HTTP/1.1 or HTTP/2.

One direction of research [CMT+17, MKM16] measures TCP and QUIC on publicly hosted websites — usually operated by Google. While this enables comparing the

performance against the same target, it lacks insights into the configuration and the load currently exhibited on the server. To enable controlled experiments, another direction [BG16, CDM15, KJC+17, NdOA+18] uses self-hosted servers. To the best of our knowledge, Yu et al. [YXY17] are the only ones that investigate the impact of packet pacing in QUIC as a tuning option — yet, they do not perform a comparison to TCP.

While self-hosting offers greater freedom, websites today are composed of a variety of resources that are often hosted by third parties on different servers. To this purpose, many studies consider websites with diverse resources but deploy only a single server [BG16, CDM15, MKM16]. The Mahimahi framework [NSD+15] was designed to replicate this multi-server nature of current websites into a testbed enabling studying realistic websites. Nepomuceno et al. [NdOA+18] perform a study with Mahimahi but find that TCP outperforms QUIC.

Another line of research investigates the performance of QUIC using (visual) Web performance metrics. Seufert et al. [SSW+19a, SSW+19b] investigate the QoE of Youtube video streaming by recording application layer metrics such as video quality or stalls and find no evidence for QoE improvements of QUIC over TCP. Rajiullah et al. [RLK+19] achieve a similar result by conducting mobile measurements using the MONROE framework. They evaluate technical metrics such as First Visual Change (FVC), Last Visual Change (LVC), and the RUM Speed Index (SI). While they find websites where QUIC has a definite impact, they conclude that overall, it has a negligible impact.

All of these works utilize gQUIC as shipped with the Chromium browser, which Google optimized for the Web. Especially the works in controlled environments use only a TCP as shipped with the Linux defaults. However, as we have seen in Section 3.1, particularly Internet giants tune their TCP stacks which challenges the comparison of a highly tuned QUIC against an untuned TCP. To this end, we perform repeatable eye-level comparisons of TCP and QUIC and are able to show that tuning TCP similar to QUIC indeed closes the performance gap.

QUIC Security. A first security analysis gQUIC's key exchange is presented in [FG14], followed by a later analysis of the complete protocol [LJB+15]. Jager et al. [JSS15] complement these works by presenting an attack vector in which the server config can be computed offline to impersonate a server.

QUIC Deployment. We further complement these works by providing the first broad assessment of QUIC usage in the wild and outside Google's network. We study both the QUIC-enabled infrastructures and its traffic shares from three vantage points. After the publication of our initial study in [RPD+18], Trevisan et al. [TGD+18] also investigate QUIC traffic shares in an ISP network. Their data partially precedes ours, but they can also show the increasing traffic shares of QUIC that are similar to ours and thus validate our measurements. Further, they also document the power of Internet giants in pushing new Internet transports.

Related but orthogonal to our work, Piraux et al. [PDB18] propose an active test suite for iQUIC that tests a variety of essential protocol features. In that regard, they can analyze how publicly announced QUIC endpoints, e.g., adapt to various

protocol versions. Unfortunately, their data ends where our data starts, and thus, we are unable to investigate how deployed infrastructure evolves in comparison to openly announced testing endpoints. We are not aware of other works that analyze QUIC deployments.

3.2.2 Measurement Overview

We start our journey into the evolution of QUIC by investigating its prevalence in the Internet before we dive into its performance. Throughout our three-year-long observations of QUIC, we have gathered several datasets that partly overlap. After slightly more than a year, in October 2017, we performed an in-depth analysis of the, at the time, current gQUIC deployment.[15] Corresponding to this one year of monitoring, we collected several traffic traces (MAWI, ISP, and IXP) for a day in August 2017.

After this day, we were able to obtain further traces from the ISP, however not from the IXP. We have nevertheless continued to monitor the general deployment of gQUIC and later also iQUIC but lack an additional in-depth analysis from a later point in time. In general, due to the limited general deployment of QUIC, we, however, performed one additional gQUIC fingerprinting study in September 2018. As we will see, few Internet giants mainly drive QUIC, thus, further probing major parts of the DNS, as we did in our in-depth analysis, puts increased pressure on the Internet as well as on our measurement infrastructure. We thus refrained from performing periodic zone-file measurements and concentrated on IPv4-only scans for gQUIC and iQUIC.

The datasets thus motivate to also split our discussions providing first an in-depth look of how QUIC evolved in a year and then how it continued to evolve before we investigate its performance.

3.2.3 An Introduction to gQUIC's Handshake

While we have discussed iQUIC in general in Section 2.3.2, we lack an understanding of gQUIC. Both protocols share many features but are vastly different in their wire image and further differ in details.

In this section, we thus introduce the *gQUIC* connection establishment phase, which as we will see shares many concepts with iQUIC. Revisiting the handshake here allows us to focus on the crucial parts that we utilize in our measurements for host enumeration and certificate grabbing. Notwithstanding, for a broader discussion of gQUIC's features and design choices, we refer to [LRW+17].

One of QUIC's main features is a fast connection establishment: In the ideal case, when cached information of a prior connection is available, it does not even take a single round-trip (0-RTT) to send encrypted application data. In the worst case (without prior connections as in our measurements), QUIC needs at least three round-trips (both IETF and Google) as shown in Figure 3.19 and explained next.

[15] At this time, iQUIC was in an early standardization stage and no deployments were known.

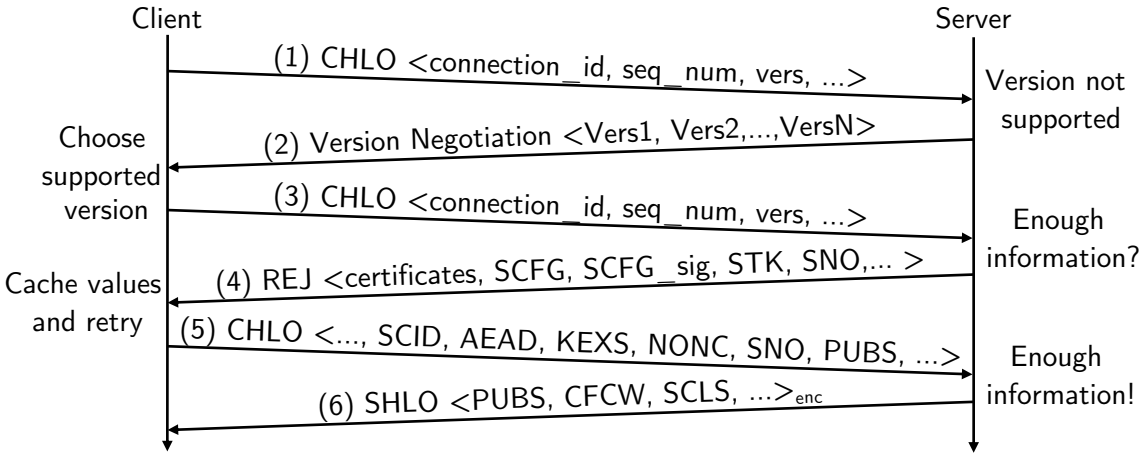


Figure 3.19 A long gQUIC handshake including version negotiation and caching of values.

In gQUIC, Clients initiate a connection using a Client Hello (`CHLO`)(1) including the QUIC version it desires to use. In case the server does not support this version, it may send a version negotiation packet (2) enabling the client to choose from a list of supported versions for a second try. We will utilize packet (1) to quickly probe for QUIC-capable hosts[16] with only a single packet exchange and analyze their supported versions provided in (2). Using a supported version, the client may advance in the handshake by sending another `CHLO` (3), without prior communication, it does not possess enough information about the server to establish a valid connection. The server supplies the necessary information (4), in one or multiple exchanges (i.e., QUIC may repeat Step (3) and Step (4) until all required data is available). In these step(s), the client will be given a signed server config (SCFG) including supported ciphers, key exchange algorithms and their public values, and among other things the certificates authenticating the host. We will utilize this information to analyze the server-provided certificates. With this information, the client can issue another `CHLO` (5) including enough information to establish a connection, the client may even send encrypted data following the `CHLO` which depicts the optimal case for a 0-RTT connection establishment. Following the `CHLO`, the server acknowledges (6) the successful connection establishment with a Server Hello (`SHLO`), containing further key/value-pairs enabling to fully utilize the connection.

3.2.4 Availability: QUIC Server Infrastructure

We start by analyzing the availability of QUIC in the Internet, i.e., how many IP addresses, domains, and infrastructures support QUIC following our in-depth dataset from 2017. Continuing, we highlight how QUIC continued to evolve in 2018 and 2019. We start by providing details about how we identify QUIC-capable hosts in IPv4.

[16]Please note the iQUIC invariant header in Figure 2.9a which also contains the version in the first packet and allows similar enumeration.

IP Scan Methodology

To quickly probe the entire IPv4 space for gQUIC and iQUIC-capable hosts, we extend ZMap [DWH13], which enables to enumerate IPv4 addresses rapidly. To identify QUIC hosts, we use QUIC's version negotiation feature (see Section 2.3.2 and Section 3.2.3). As QUIC is build to enable rapid protocol development and deployment, negotiation of a supported version (i.e., supported by client and server) is fundamental to its design. That is, the protocol requires announcing a version identifier in the initial packet sent from the client to the server. In case the version announced by the client is not supported by the server, it sends a version negotiation packet. This packet lists all supported versions by the server, enabling the client to find a shared version that they use in a subsequent handshake.

We leverage this feature and sent a valid handshake message containing a version that is likely to be *unsupported* by the other party, i.e., by including a version that is not reserved and does not follow the current gQUIC pattern. For iQUIC, we do the same but use a version following the pattern `0x?a?a?a?a` (filling `?` with random bytes), these versions are specifically reserved to be never used and thus exercise version negotiation. To allow a curious observer to identify and reason about our scanning, we use the connection ID fields in iQUIC to transmit an ASCII text that reads `Visit:quic.netray.io`, announcing a website that presents more information on our scans. For gQUIC, we are limited in size and transmit the word `SCANNING` as the connection ID again hinting at our intentions. Regardless of QUIC-flavor, in response to our initial packet, the server will not be able to continue the handshake as both versions do not match, and thus, it will send a version negotiation packet containing a list of its supported versions. Using an invalid version or reserved version has the advantage that we enumerate not only valid QUIC hosts but also gain further insights about the server, namely the list of its supported versions. We declare an IP address as QUIC-capable if we either receive a valid version negotiation packet or, in gQUIC, a public reset packet (comparable to a TCP RST). We build and publish [Rüt19] ZMap modules implementing this behavior enabling rapid enumeration of QUIC hosts in the IPv4 space.

3.2.4.1 gQUIC Census in October 2017

gQUIC Hosts. Figure 3.20 shows that the total number of gQUIC-capable IP addresses (sum of the stacked area) has more than tripled from 186.77K IP addresses in August 2016 to 617.59K IP addresses in October 2017. As of October, we find IP addresses in 3.04K ASes. To analyze who drives this trend, we attribute gQUIC IP addresses to providers: we classify IP addresses by *i)* AS information, *ii)* per-IP X509 certificate data (e.g., who issued the certificate, who owns it), and *iii)* per-IP rDNS data (e.g., Akamai configures rDNS entries such as `*.deploy.static.akamaitechnologies.com`), using data available at Routeviews and scans.io. As of August 2016, we can already attribute 169.52K IP addresses to Google. They have since doubled their gQUIC-capable infrastructure to 330.62K IP addresses as of October 2017, accounting for 53.53% of all gQUIC-capable IP addresses. We identify Akamai as the second-largest gQUIC-enabler: they started to

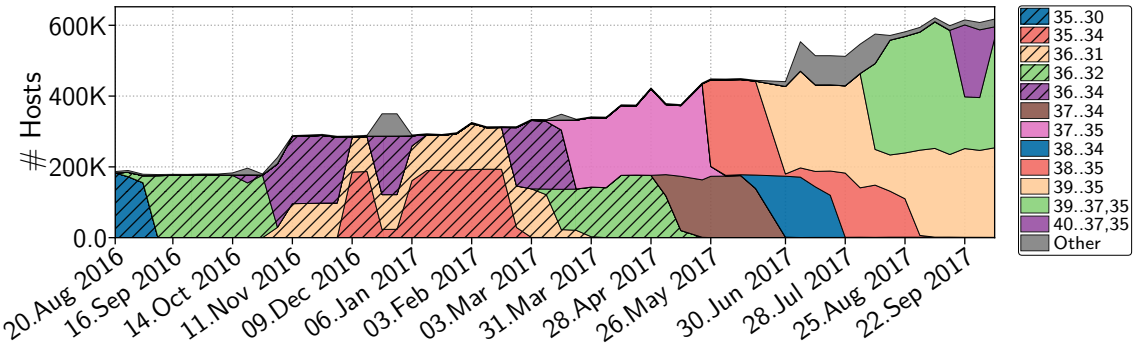


Figure 3.20 Number of gQUIC-capable IP addresses and support for sets of certain gQUIC versions, here we display versions when there was support by at least 20 000 hosts once. Versions that first appeared in 2016 are hatched.

increasingly deploy gQUIC on their servers in November 2016, while we find around 983 Akamai IP addresses in August, the number jumped to 44.47K IP addresses in November 2016. Akamai has since then continued to deploy QUIC having 251.43K IP addresses as of October 2017 accounting for 40.71% of all gQUIC-enabled IP addresses.

To classify the remaining 35.54K hosts, we executed TCP HTTP GET / on port 80 for these IP addresses. However, for 23.91K IP addresses, we could not get any data due to i/o timeouts. Apart from this, we find 7.34K hosts announcing a *LiteSpeed* server string, a Web server that added gQUIC support in mid of July 2017 [Lit17]. We find servers announcing *gws* (1.69K) and *AkamaiGHost* (1.44K), hinting at even more Google and Akamai installations. The fourth-largest group of servers announces *Caddy* (356) as the server string, this server uses the quic-go [Cle19] library and can also be used as a reverse proxy for other TCP-only servers.

Takeaway. *We observe a steady growth of gQUIC-capable IP addresses, mainly driven by Google and Akamai. Few IP addresses already use third-party server implementations.*

gQUIC Version Support. Since gQUIC is under active development, it requires clients and servers to be regularly updated to support recent versions. To understand how the server infrastructure is updated, Figure 3.20 shows the number of hosts supporting a specific set of versions (recall: A host may support multiple versions!). The figure shows that many version combinations have a short lifespan in which old versions fade away, and new versions appear. For example, hosts supporting version `Q035` down to version `Q030` switch to versions `Q036, ..., Q032`, thus losing support for two versions. While some versions fade away, we also see that, e.g., version `Q035` is supported by almost all hosts throughout our dataset. Even though, to the end of this observation period support for version `Q036` is dropped. While this shows that some versions offer long-term support, the figure also shows how vibrant the gQUIC landscape is even though a single Internet giant, Google, mainly push its development.

Given that some versions introduce radical protocol changes without backward compatibility, questions concerning the long-term stability of a QUIC-Internet arise.

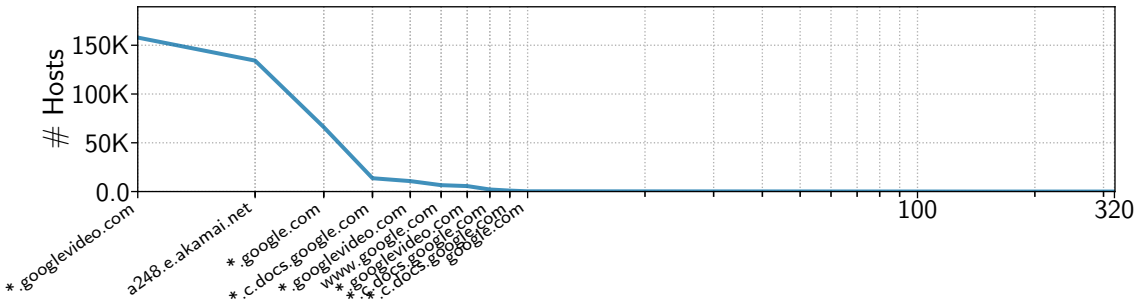


Figure 3.21 Number of hosts giving out the same certificate on the y-axis. First listed common names for the 10 certificates with the highest coverage shown on the log x-axis.

On the one hand, the ability to easily update the protocol offers the possibility to introduce new features and thereby to evolve the protocol quickly. On the other hand, updating Internet systems is known to be notoriously hard. The vast number of legacy systems raises the question of long-term compatibility — designing implementations to be easy to update is challenging.

Takeaway. *gQUIC is subject to rapid development reflected in frequent version updates. Given its realization in user space at the application-layer, this property is likely to stay: future transports can be potentially updated as frequently as any other application. This observation motivates future measurements to assess the potentially highly dynamic future Internet transport landscape.*

Enumerating gQUIC Domain Names

Methodology. We develop a second tool that finishes the handshake and enables to further classify previously identified hosts and infrastructures. By accounting for the mandatory SNI, it can present a hostname that is necessary for the server to deliver correct certificates when hosting multiple sites on a single server. We base our tool [Rüt19] on the quic-go [Cle19] library, which we extended to enable tracing within the connection establishment to extract all handshake parameters (see Figure 3.19).

IP-based Certificate Scan. In a first step, we cluster all previously discovered gQUIC-enabled IPs by their X509 certificate hash. This step enables us to better understand gQUIC-enabled infrastructures. Since the server's hostname is unknown at the request time when enumerating the IPv4 address space, we present dummy domains (e.g., foo.com) to each IP address and retrieve the X509 certificate. The retrieved certificate provides information on the domain names for which the certificate is valid, which can indicate the hosting infrastructure. We remark that this approach yields the *default* website that is configured at a server and will not identify different sites in the presence of SNI. We find that 216.64 K hosts require SNI and do not deliver a certificate (for which we account for when scanning domain zones later). Figure 3.21 shows that we only observe 320 different certificates for the probed 617.59 K gQUIC IP addresses. The heavy-tailed distribution shows the top-five (ten) certificates already represent 95.41% (99.28%) of the IP addresses, most prominently

	06. Oct 2017 .com		03. Oct 2017 .net		04. Oct 2017 .org		08. Oct 2017 Alexa 1M	
# Domains	129.36 M	(100.0%)	14.75 M	(100.0%)	10.37 M	(100.0%)	999.94 K	(100.0%)
gQUIC-enabled	133.63 K	(0.1%)	8.73 K	(0.06%)	6.51 K	(0.06%)	11.97 K	(1.2%)
Valid Certificate	2.14 K	(0.0%)	181	(0.0%)	159	(0.0%)	342	(0.03%)
Timeout	114.63 M	(88.61%)	10.80 M	(73.23%)	8.09 M	(78.06%)	826.67 K	(82.67%)
Version-failed	29	(0.0%)	6	(0.0%)	1	(0.0%)	5	(0.0%)
Protocol-error	606	(0.0%)	222	(0.0%)	0	(0.0%)	1	(0.0%)
Invalid-IP	322.24 K	(0.25%)	59.24 K	(0.4%)	40.15 K	(0.39%)	15.42 K	(1.54%)
DNS-failure	13.76 M	(10.64%)	2.40 M	(16.26%)	1.18 M	(11.41%)	49.34 K	(4.93%)

Table 3.7 gQUIC support in different TLDs and in the Alexa Top 1M list.

Google and Akamai. We validated that these IP addresses truly belong to both companies by requesting content via TCP and HTTP on port 80 on the same hosts. We next assess gQUIC support among domain names.

Probing Complete Domain Lists. Presenting a non-existing SNI name in our previous measurement will miss any server that enforces to present a valid hostname. Thus, we next assess the gQUIC support by probing complete domain name lists. That is, we probe all domains in the .com/.net/.org zone files and the Alexa Top 1M list. These zones are available at Verisign [Ver19] (.com/.net) and PIR [Pub19] (.org). Together they contain more than 150M domains, i.e., about 46 % of the domain space [Ver17]. We use zDNS to resolve the domains, and for each successful resolution, we use our tool to check for gQUIC support and to grab all parameters from the connection establishment. The whole process takes roughly 15 h and is thus feasible to run daily. However, as gQUIC CHLO packets require padding, they nearly fill the MTU; the scan saturates a 1 Gbit/s link easily.

Table 3.7 shows the gQUIC-support in the .com/.net/.org zones as well as in the Alexa Top 1M list. We define *gQUIC-enabled* domains as being able to initiate a gQUIC handshake. A domain is tagged as *Timeout* when we received no response to our initial gQUIC `CHLO` within 12 s, e.g., in the absence of gQUIC support. We furthermore show some specific errors as well as *DNS-failures*.

Overall gQUIC-support is very low. Depending on the zone, 0.06% – 0.1% of the domains are hosted on gQUIC-enabled hosts. Only 1.6% – 2.44% of these domains present a valid X509 certificate. This questions how many domains really deliver content via gQUIC.

Landing Page Content. Websites can utilize different server configuration and even different server implementations for different protocols. The successful establishment of gQUIC connections does thus not imply that meaningful content is being served. To assess how many gQUIC-capable domains deliver content similar to their TCP/HTTP counterparts, we instruct Google's gQUIC test client (part of the Chromium source) to download their landing page via gQUIC. We then compare their content to their HTTP 1.1/2 counterparts, which should be similar if these gQUIC-capable domains are correctly set up. We disabled certificate checks to probe all capable domains.

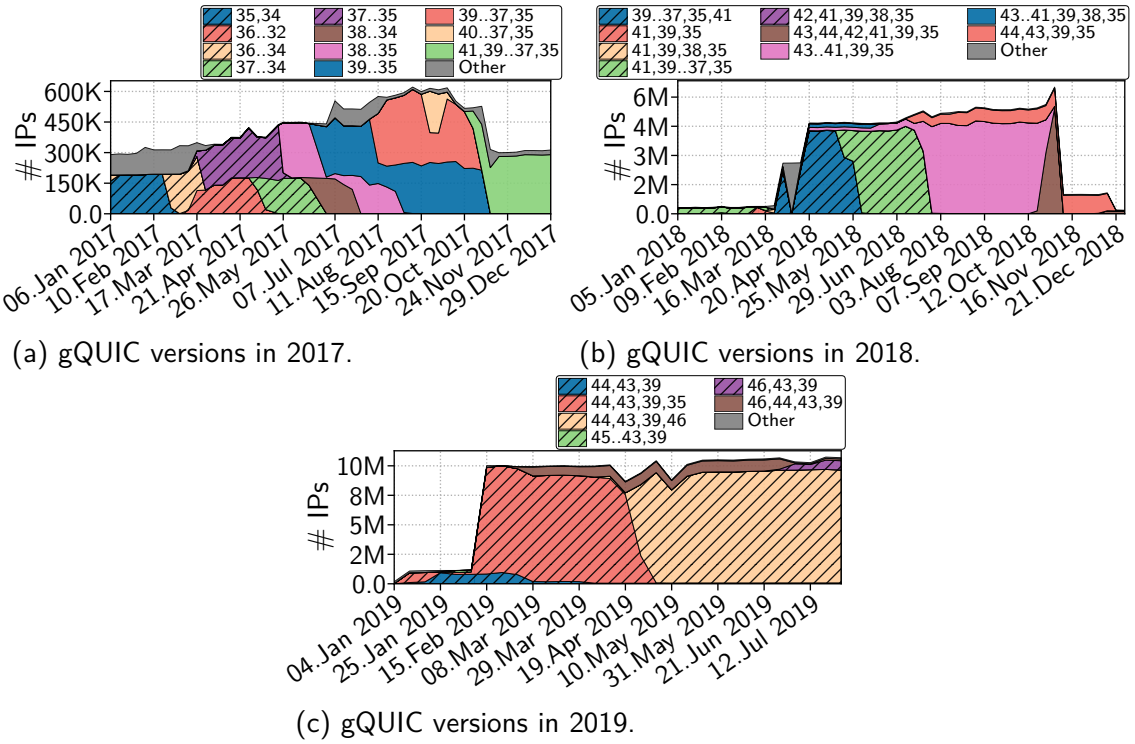
Out of the probed 161K domains, 16K (9.8%) return no data and 33K (20.7%) >1 kB via gQUIC. In the case of the latter, 33K domains (22K served by Akamai) do deliver content similar to their TCP/HTTP counterpart. We define similarity by structural Hypertext Markup Language (HTML) similarity (e.g., in the number of tags, links, images, scripts, ...) and require > 3 metrics to agree to define a Web page to be similar. Domains delivering similar content over gQUIC are thus in principle ready to be served by a gQUIC-capable browser. To be discovered by a Chrome browser, they, however, need to present an alternative service (`alt_srv`) header via TCP-based HTTPS pointing to their gQUIC counterpart. 11K domains present this header via HTTPS (5K hosted by Google and 0 by Akamai) and only 7 via HTTP. Thus a large share of the domains would not be contacted by a Chrome browser even though gQUIC support is in principle available. The header further specifies the gQUIC versions supported by the server, of which at measurement time Chrome requires gQUIC Version 39. Only 5K domains present this version in their `alt_srv` header, all hosted by Google. We remark that our content analysis only regards *landing* pages and does not account for additional assets (e.g., images or videos). Usually, CDNs offer dedicated products for media delivery, whose gQUIC support could differ. Assessing their gQUIC support in detail, thus provides an interesting angle for future work.

Takeaway. *The limited number of X509 certificates retrieved in our IP-based scan hints at the small number of different providers currently using or experimenting with gQUIC. Furthermore, only a small fraction of the monitored domains are hosted on gQUIC-capable infrastructures — an even smaller fraction can actually deliver valid certificates for the requested domains. Regardless of the certificate, many gQUIC-enabled domains do deliver their pages via gQUIC. Still, in our measurements, many would not be contacted by a Chrome browser, either because of a non-present* `alt_srv` *header or insufficient version support. There is thus a significant potential to increase gQUIC support.*

3.2.4.2 Evolution of gQUIC in 2018 and 2019

We complement our in-depth census from October 2017 with a continuous view on the evolution of gQUIC-capable hosts in IPv4 fueled by our IP address-based scans. Figure 3.22 visualized the evolution in 2017, 2018, and 2019.

As shown in Figure 3.22a, right after our census in October 2017, we noticed a sudden drop in gQUIC support in November 2017. At first, about half of all gQUIC IP addresses no longer responded, which we could largely attribute to Akamai. While we received ∼5K public gQUIC RST packets in October 2017 to our probes (similar to a TCP RST), they spiked to over ∼76K until the 17th of November, 2017 when they returned to ∼5K. Moreover, we found that gQUIC Version 40 was decommissioned. Later the Google gQUIC source code documents that Version 40 was an attempt to move to the IETF frame format. However, a bug in the implementation caused it to never ship [Chr18c]. Surprisingly, we nevertheless find gQUIC Version 40 for roughly a month. Whether or not the sudden disappearance of Akamai IP addresses is related to Version 40 is not known to us. For 2017, we do not find them to reappear.



(a) gQUIC versions in 2017.

(b) gQUIC versions in 2018.

(c) gQUIC versions in 2019.

Figure 3.22 Number of gQUIC IP addresses announcing a certain set of versions in 2017/2018/2019 (shown when at least 150k hosts use this version). Note the different y-axis scales. This also leads to the seemingly visual stability in (c) even though the total amount still varies as before.

Looking at Figure 3.22b, we observe this does continue for the first quarter of 2018 with only again a version dropout in March.

Akamai to Push gQUIC. At the 30th of March, 2018, we observe a massive increase (note the different axis scaled from 2017 and 2018) in gQUIC-capable IP addresses and a new version set to appear. For the first time, we observe that hosts announce the most recent version as the last version. By again looking at rDNS entries and routing information for these IP addresses, we find that they belong to Akamai. At the 10th of April, 2018, we find a corresponding Akamai blog entry [AC18] explaining that Akamai will ramp up gQUIC as of the 1st of June, 2018. On our scan on the 20th of April, we find another massive increase to over 4M gQUIC-capable IP addresses. We observe that Akamai switches the announced versions for a part of their servers in May and entirely as of the beginning of June to announce Version 41 as the first version. Since Chrome will currently only use a single Google-dictated gQUIC version, we believe this to have been a measure to deploy the infrastructure but to control who will connect. As of July of 2018, we again find a slight increase in gQUIC-capable IP addresses and the appearance of gQUIC Version 44 which implements the (at the time current) IETF header format [Chr18b]. In November 2018, we again observed a significant drop in gQUIC-capable IP addresses. Only in February 2019, they reappear but this time massively with a new all-time high of nearly 10M IPs, with roughly 6M IP addresses directly hosted in Akamai ASes.

Measurement Challenges. Providing long-term measurements is a challenging task. We have configured our infrastructure to scan every week such that a scan completes in roughly a single day. Especially, after having set up the scan, it is unfortunately not enough to only wait to gather enough data. During our measurements, we had several incidents that required manual intervention, ranging from hard drive failures to protocol changes. For example, in late December 2018, further gQUIC-capable hosts disappeared which we investigated in January 2019, we found that Google changed the required minimum packet length. In turn, we adjusted our measurement tools to account for this. At the same time, we changed which version we are announcing to follow the reserved versioning scheme introduced with iQUIC. Further, for no particular reason, we also changed the packet number length from a single byte to two bytes. In March 2019, we were contacted that we were missing some servers that should be available for gQUIC. After investigating and discussing with the contact, we found that our change to two bytes (even though compliant with Google and covered in the gQUIC drafts) was incompatible with Litespeed gQUIC. Consequently, we switched back to using one byte. While such changes are minimal, it can take time to understand their gravity, especially when dealing with large numbers of results on an Internet-scale. Further, an actively developed protocol like QUIC demands constant attention to follow the current line of development.

Takeaway. *Our measurements reflect the enormous efforts that are put in place to make QUIC a standardized Internet reality. The observations highlight the deployment challenges of a new Internet transport, and current efforts in experimenting with gQUIC highlight the readiness of major Internet stakeholders once QUIC finishes standardization. As of August 2019, we find Akamai to have the largest gQUIC deployment in IPv4, followed by Google.*

gQUIC Parameterization & Active Fingerprinting

In September 2018, we again performed a comprehensive evaluation of gQUIC. This time, we focused on gQUIC connection parameterization and if this offers the potential for fingerprinting server software, hoster, or service. Since QUIC mandates SNI, our IPv4 data lacks domain names to establish valid connections. We thus use large domain name lists of > 180 TLDs and perform DNS resolutions from our university network and subsequently attempt to connect with our quic-go QUIC client (see Section 3.2.4.1) to gather connection parameters exchanged during the handshake. In total, we investigate over 180 zones, the largest in our scans are again the .com/.net/.org zones.

Connection Parameters. We find ∼218K hostnames mapping to ∼19K unique gQUIC-capable IP addresses residing in 659 different ASes. Over the course of the handshake, gQUIC allows exchanging a list of arbitrary (tag, value) pairs (see Step (4) in Figure 3.19), e.g., the stream flow control rwnd (SFCW) tag's value denotes the initial stream flow control receive window. We find 32 different tags used in the handshakes. Of these, we find ten groups with different occurrences as visualized in Figure 3.23. The gQUIC sources do not even define all of these tags, e.g., the server ID (SVID) seems to be exclusively used by the quic-go implementation to signal the implementation similar to the HTTP server header field.

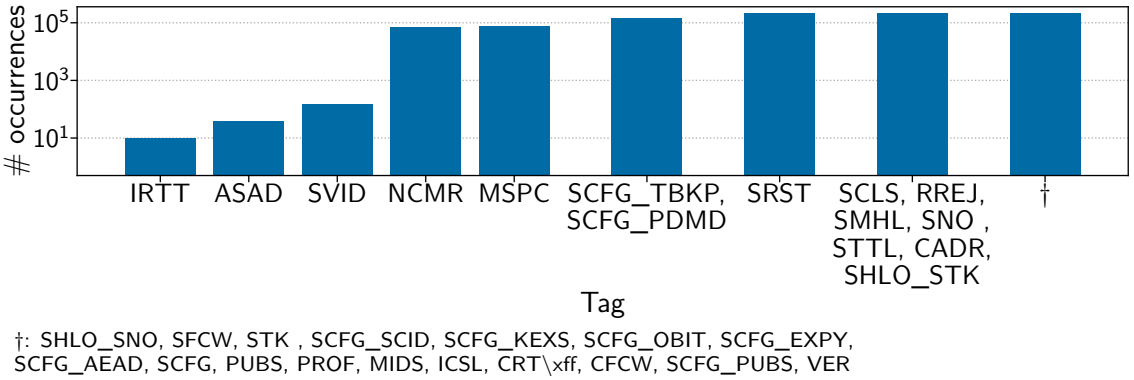


Figure 3.23 Different groups of tags having the same number of occurrences and their number of occurrences in our handshakes. Note the log scale.

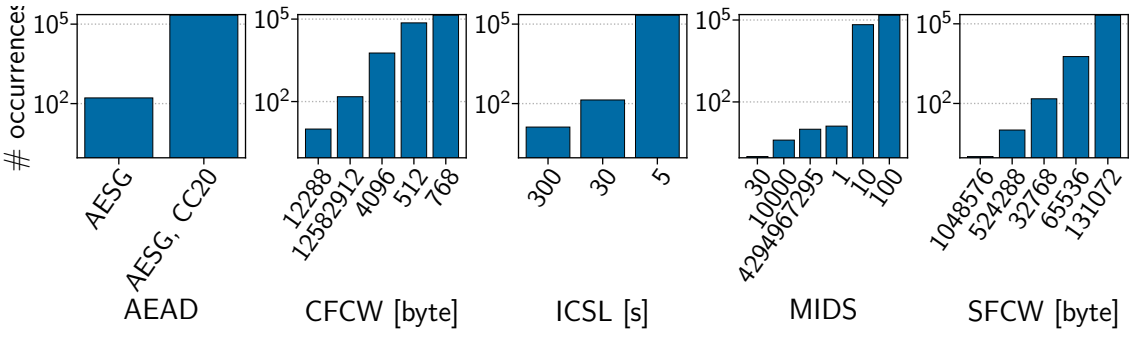


Figure 3.24 Tags and their observed values (x-axis) and how often we found them in the handshake (y-axis)

Transport Parameters. We focus on mandatory parameters that are required to exchange data and thus are present at all hosts (†). Some of these tags such as the source-address token (STK) authenticating the requesters IP address (similar to a TCP Fast Open cookie) are unique, and we thus focus on five elementary parameters: the authenticated encryption with associated data (AEAD) contained in the SCFG, the initial connection flow control receive window (CFCW), the idle connection state lifetime (ICSL), the maximum incoming dynamic streams (MIDS), and the initial SFCW.

Figure 3.24 shows the values of the tags and how often we found these values in our handshakes. Currently, only two different cipher suits (AEAD) are defined, most hosts seem to support both, and only a small set of hosts support only one. The CFCW is comparable to TCP's initial rwnd for the whole connection. We find values from 512 B to 12 MB hinting at the required buffer sizes. Smaller values seem to prevail. Regarding the ICSL, we find three different idle timeouts, i.e., after which the implementations close the connections when they receive no packets. Surprisingly, we find that 5 s seem rather short; however, hosts could use ping frames to keep the connections alive thus also putting pressure on a client to refresh a possible NAT binding. For this reason, 300 s and even 30 s could lead to losing the binding. Most servers set low values for the MIDS. Currently, most gQUIC servers seem to adopt the HTTP/2 recommendation of 100 streams [RFC7540]. Apart from this, we find ten and 30 as well as much larger values which could be application-specific settings.

	Google	Caddy	Trancent Cloud	Verizon CDN	Unreach	AK1	AK2	YouTube 8.8.8.8	Google Beacon	GMail
CFCW	768	12582912	12288	4096	12582912	4096	512	768	768	768
ICSL	5	30	5	5	300	5	5	5	5	5
MIDS	100	100	4294967295*	100	1	100	100	10	10000	30
SFCW	131072	32768	524288	65536	32768	1048576	131072	131072	131072	131072
Count	71486	137	10	5837	13	1	72896	67427	4	1

*: unsigned int32 max

Table 3.8 gQUIC infrastructures clustered by k-Means.

The SFCW values are significantly larger than those for the whole connection. Here we find values that seem typical for TCP rwnds in various operating systems (see Figure 3.4 on Page 70 from our previous contribution).

Fingerprinting. Given this wide variance in the parameters that we observed, we want to find out if they offer potential to fingerprint, e.g., server implementations, CPs, or services. While a tag such SVID can be directly used to derive a server's implementation, it is well known that connection parameters can be used to fingerprint different transport protocol implementations such as in p0f [Zal14]. To investigate the fingerprinting potential, we focus on the four integer-based tags — CFCW, ICSL, MIDS, and SFCW — and apply a k-means clustering. We increased k until we could not derive any new classes and arrived at $k = 10$ classes contained in our data, which we summarize in Table 3.8. The table lists the configuration of the four tags and the number of hosts that each class contains.

Subsequently, to derive a ground truth, we labeled each class by inspecting the IP address AS ownership, DNS redirections, patterns in the hostnames, and server software used on TCP+HTTP. For large classes, we investigated a random sample, and for smaller classes, we verified the random sample on all elements in the class. The *Google* class accommodates server on Google infrastructure; from our random sample, we found many hosts to contain Google-specific URL parts or IP addresses belonging to Google. The *Caddy* class contains hosts that use the Caddy [HLA18] Web server, a Web server with gQUIC support, which we found with the help of the HTTP server string. According to the hostname or ASN, Tencent [Ten18] hosts the next class (*Trancent Cloud*), i.e., the company responsible for WeChat. Via DNS CNAME redirects we find the *Verizon* class which contains hosts residing in Verizon's infrastructure. For 13 hosts in the *Unreach* class, we are unable to specify a proper label since we are unable to reach any of these hosts via TCP nor are any other investigations conclusive. *AK1* and *AK2* are both hosted by Akamai, as again indicated via their CNAME, however, the *AK1* class contains only a single host. The *YouTube* class is rather odd, we find tons of hosts postfixed with `googlevideo.com`, but also many other hostnames whose DNS resolves to Google's public DNS recursor (8.8.8.8). Interestingly, the *YouTube* class defines only ten streams (MIDS), which seems sensible because video delivery does not require many streams. The four hosts in the *Google Beacon* class all follow the same naming scheme in the hostnames

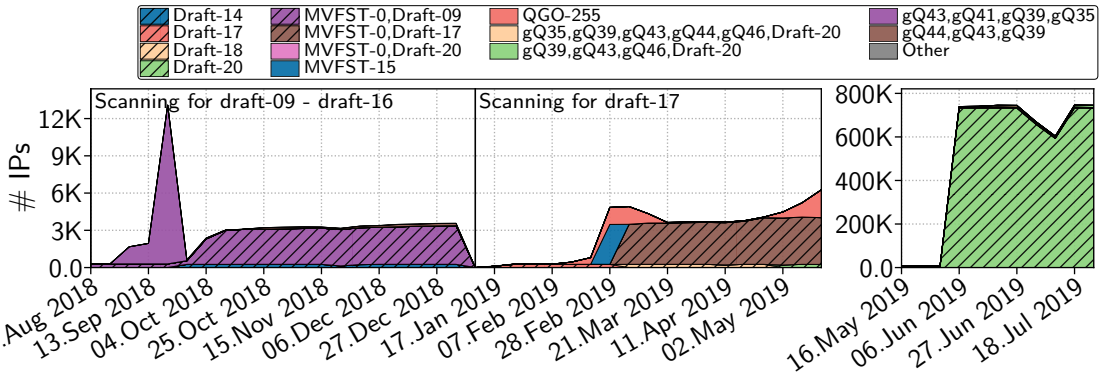


Figure 3.25 Number of iQUIC IP addresses announcing a certain set of versions (excluding reserved versions) in 2018 and 2019. Only displayed if the version set appeared at more than 25 IP addresses. Annotations highlight when our scanner switched to a wire format version.

containing the word beacon. Here the opposite is visible, a large number of streams is possible. The last *GMail* class contains a single host which we found through the hostname `mail-attachment.googleusercontent.com`.

Takeaway. *gQUIC offers a versatile environment to encode various connection-specific parameters. We have only peaked in the data and have found a variety of different configurations and parameters. By looking only at four parameters, we were able to automatically device meaningful classes for different implementations, hosters, or services, thus showing potential for extensive fingerprinting — an exciting angle for future work.*

3.2.4.3 The Rise of iQUIC

We now shift our focus away from gQUIC and investigate the rise of iQUIC as currently standardized by the IETF. To do so, we rely on the same mechanism that we already exploited for gQUIC, i.e., the version negotiation. However, as gQUIC and iQUIC are incompatible, we create a separate ZMap module for scanning iQUIC.

2018. As shown in Figure 3.25, we scheduled our first scans for iQUIC in August 2018. Back then, we found little support for iQUIC, and only Facebook's MVFST implementation with roughly 300 servers was publicly available. We observed the largest peak of iQUIC compliant version negotiation in September 2018 with roughly 12k IP addresses, yet, announcing gQUIC versions. We find that the IP addresses belong to Verizon's Edgecast CDN; however, they soon after disappeared from the public Internet. Over the remainder of 2018, we observe that Facebook expands their iQUIC tests, and over 3k IP addresses are reachable. Since we use the information published via the IETF QUIC working group's wiki[17] to decode versions, we were surprised to see QUIC Version 101, i.e., a version that at that time was supposed to be reserved for ratified standards. After bringing this issue up on the QUIC IETF mailing list, we learned that a development version of QUIC-GO accidentally uses this version which was then subsequently changed by the authors. Nevertheless, we

[17] `https://github.com/quicwg/base-drafts/wiki/QUIC-Versions`

were unable to figure out who operated the IP addresses as they were hosted in the Amazon or Google cloud, resided in Chinese ASes and only one IP address hinted at Alibaba.

2019. At the beginning of 2019, we switched to iQUIC draft-17[18] whose wire format is fundamentally incompatible with the previous drafts. We found only 13 IP addresses of which 12 were still announcing QUIC-101 and one IP address by Apple that announced draft-14 even though that draft is not compatible with draft-17 wire format. One week later, we observe an increase of over 100 IP addresses supporting draft-17 hosted by Cloudflare that further increased in the weeks after. As of February, draft-17 compliant quic-go (QGO-255) versions appear of which Alibaba hosts the majority of the IP addresses. At the end of February and in the first weeks of March, we see a surge of Facebook-hosted IP addresses to appear again that at first announce MVFST Version 15 and then switch to announcing MVFST Version 0 as well as Draft-17 itself. As of the 14th of March, draft-18, that the IETF released in late January 2019[19], is announced by Cloudflare-hosted IP addresses, by a single WinQuic implementation by Microsoft (not shown in the figure) as well as by a single version also announcing gQUIC versions (also not shown). We observe the most substantial increase in iQUIC-capable IP addresses in June 2019, which is again driven by Cloudflare. To the end of our observations, we find over 745k iQUIC-capable IP addresses.

Version Greasing. iQUIC recommends adding random reserved versions to the version negotiation packet, i.e., this *greasing* helps that middleboxes do not ossify around parsing a particular fixed version in the packet at a well-known offset as well as that clients are forced to be able to handle unsupported versions from the start. As of the 14th of March 2019, we found 4346 iQUIC IP addresses, of these only 762 greased the version negotiation field. However, this is mostly due to Facebook's implementation that contributes 3326 IP addresses not performing this greasing as well as Cloudflare not doing it. Apart from that we, e.g., observe roughly half of all quic-go implementations to have the greasing in the first version and the other half having it in the last version (currently they only announce one non-reserved version). As of July 2019, the situation remains unchanged, and Cloudflare as the largest iQUIC-supporter does still not grease. So, in summary, it seems that most implementations already grease their version negotiation, but some large players still lack this functionality.

Measurement Challenges. One of the challenges during our measurements was to keep up with the changes made to the drafts. While we were able to trigger version negotiation for multiple drafts with a request made for one specific version, we nevertheless had to implement the changes to the headers to properly validate the different responses. Unfortunately, when having to perform a hard switch, e.g., as to draft-17, one would need to double the effort to monitor the use of both flavors of the protocol. While we do this for gQUIC and iQUIC, we felt that doing so for iQUIC twice would mainly increase traffic and abuse while not offering many new insights. However, as also visible in Figure 3.25, we are, e.g., unable to reasons if

[18]Draft-17 appeared in mid December 2018

[19]compatible with draft-17 version negotiation

Facebook shut down their servers as of January, e.g., due to a bug or just waited to transition to draft-17. Another strategy would have been to continue probing the known IP addresses to follow up on their changes that are out of scope for the general IPv4-wide probing.

Takeaway. *While iQUIC servers do appear in IPv4, their share is far behind gQUIC. However, we observe a much larger pool of players that seem to experiment with iQUIC than we observed for gQUIC. Thus, once iQUIC leaves its initial standardization, and major browsers support it, it seems likely that there will be a more diverse QUIC landscape than before. However, this may in the end challenge the rollout of new versions while deprecating others when QUIC is also integrated in other products besides browsers and Web servers.*

3.2.5 Usage: QUIC Traffic Share

We complement our view on QUIC infrastructure with an extended look on QUIC traffic shares. To this end, we again take an in-depth look at traffic shares in 2017. Subsequently, we analyze several additional ISP traces one year later.

Traffic Classification

We use protocol and port information to classify HTTPS (TCP port 443), HTTP (TCP port 80), and QUIC (UDP port 443). We chose this classification since it applies to all of our traces: MAWI (PCAP header traces) and ISP + IXP (Netflow traces without protocol headers). Furthermore, QUIC is hard to identify even in PCAP traces as, by design, it is fully encrypted and, e.g., iQUIC exposes only a single bit that is useful for classification, thus having a significant potential for misattribution. We nevertheless remark that our classification can *i)* miss protocol traffic on non-standard ports and can *ii)* wrongly attribute other traffic on the monitored ports. Regardless of the accuracy, this uninspectable UDP traffic still impacts network operators as they cannot rely on methods developed for TCP to analyze the efficiency of their network. This generality also means that it captures both gQUIC and iQUIC when running on port 443. Thus, our classification reports an upper bound on the protocol utilization on standard ports.

3.2.5.1 QUIC Census 2017

Complementing our QUIC IP census in October 2017 (see Section 3.2.4.1), we now regard traffic shares that correspond to the previously presented period. Thus, restricting our observations to this time enables comparing available infrastructures and traffic shares.

We quantify the QUIC traffic share by analyzing three traces representing different vantage points: *i)* 9 months of traffic in 2017 on a transit link to an upstream ISP (MAWI dataset [MAW18]), *ii)* one day in August 2017 at a European Tier-1 ISP, representing edge (DSL + cellular) and backbone traffic, and *iii)* the same day at

	Overall				Operator's share			Share in Protocol		
	HTTP	HTTPS	QUIC		HTTP	HTTPS	QUIC	HTTP	HTTPS	QUIC
MAWI	28.0%	44.9%	6.7%	-		-			-	
ISP	37.7%	40.1%	7.8%	Akamai	67.9%	32.1%	0.1%	27.2%	12.6%	0.1%
				Google	1.4%	59.5%	39.1%	0.7%	28.8%	98.1%
Mobile ISP	24.8%	55.4%	9.1%	Akamai	57.7%	42.3%	0.0%	28.5%	9.6%	0.1%
				Google	1.6%	64.4%	34.0%	1.8%	29.5%	96.9%
IXP	32.2%	30.9%	2.6%	Akamai	33.3%	33.3%	33.3%	5.0%	5.2%	59.9%
				Google	3.1%	70.0%	26.9%	0.3%	7.2%	33.1%

Table 3.9 Average traffic shares (overall), among the operators, and the protocols. Operator's share is, e.g., from all of Google's traffic, the share of the QUIC traffic at a vantage point. Share in protocols denotes the traffic share of a protocol at a vantage point, e.g., the amount of Google's QUIC traffic from all other QUIC traffic.

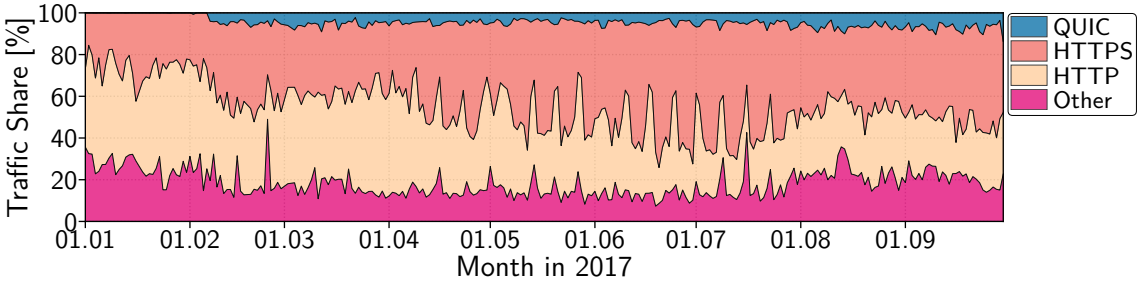


Figure 3.26 Traffic share of QUIC compared to HTTP and HTTPS in the MAWI trace.

a large European IXP. We show the per-trace traffic shares in Table 3.9, which we discuss next.

MAWI Backbone Trace. We start by analyzing traffic on a trans-Pacific WIDE backbone link provided by the MAWI working group [MAW18]. We analyze anonymized header traces available at the MAWI repository (*samplepoint F*). The monitored link is a transit link connecting the WIDE backbone to an upstream ISP. The traces involve 15 minutes of traffic captured at 14:00 h on each day. Each packet is capped to the first 96 B.

We begin to analyze traffic on the 1st of January, 2017, since Google enabled gQUIC for all of its Chrome and Google-developed Android App users in January 2017 [LRW+17]. Figure 3.26 shows the traffic volume until the end of September 2017. The trace shows that the QUIC traffic share is 0.0% in January, in contrast to the Google report of having widely enabled gQUIC in January. This abstinence suggests that the monitored user-base is not using Google products (e.g., Chrome) at the time, QUIC has not been enabled for this network, or that traffic is routed differently. We observe the first QUIC traffic in February, where the QUIC traffic share is at 3.9%. It continues to increase to 5.2% in March and reaches 6.7% in September. QUIC offers an alternative to TCP+TLS, which is the foundation of legacy HTTPS, its share is at around 44.9%, even the unencrypted version HTTP is still at around 28.0%. As the provided trace anonymizes destination and source

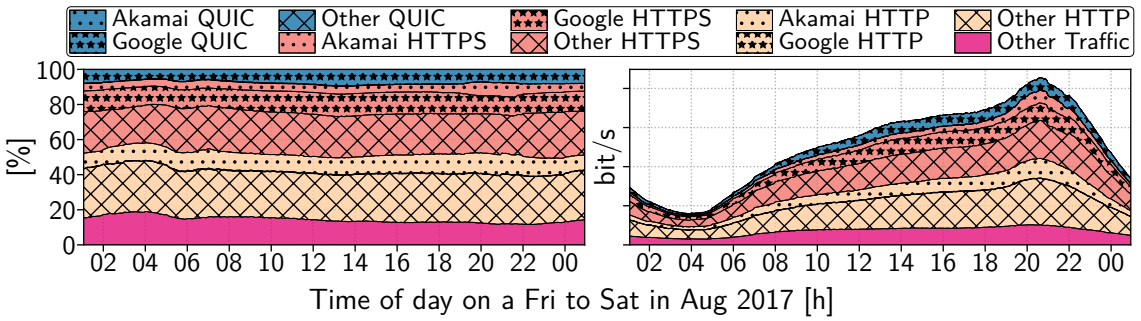


Figure 3.27 QUIC traffic share in a major European ISP (up- and downstream). Left, relative share of QUIC. Right, total traffic compared to HTTP(S), y-axis has been anonymized at the request of the ISP. Nearly all QUIC traffic is served by Google.

addresses, we cannot attribute this traffic to infrastructures (e.g., Google or Akamai) or services (e.g., YouTube) which we found to have the largest server installations. We leave this analysis to the ISP trace for which we have AS-level information available.

Takeaway. *Within nine months after its general activation by Google, QUIC already accounts for a non-negligible traffic share, demonstrating its ability to evolve Internet transport.*

European Tier-1 ISP. We obtained anonymized and aggregated Netflow traces from all border routers of a large European ISP for one day in August 2017. The Netflow traces were aggregated to 5-minute bins and all IP addresses were replaced by the corresponding ASN before they were made available to us. Thus the traces do not reveal the behavior of individual users. The captured traffic contains *i)* edge traffic by DSL, *ii)* cellular customers, and *iii)* transit backbone traffic.

Figure 3.27 shows the traffic volume (up- and downstream) for 24 h by protocol and prominent infrastructures (we removed the traffic volume (y-axis) at the request of the ISP). As our previous host-based analysis (see Section 3.2.4.1) showed that, in 2017, mainly Akamai and Google support QUIC, we also show their traffic shares (according to their ASNs). At first, we observe that QUIC traffic follows the same daily pattern as HTTP and HTTPS. On average, QUIC accounts for 7.8% of the traffic with a standard deviation of σ: 1.0%. This deviation is similar to HTTP (σ: 1.2%) and HTTPS (σ: 1.4%), which account for 37.7% and 40.1% of the traffic, respectively.

Google almost exclusively supplies the observed QUIC traffic: They account for 98.1% of the overall observed QUIC traffic. Among all of Google's traffic, 39.1% is using QUIC (σ: 2.3%), peaking at 42.1%. This share is a larger than the global average of 32% reported by Google in November 2016 [LRW+17]. In 2017, Google-developed applications (e.g., Chrome or the Youtube Android app) primarily supported QUIC. In the absence of QUIC libraries, third-party support is low (e.g., at that time Opera had optional QUIC and Firefox had no QUIC support). The availability of QUIC libraries thus has the potential to improve client support drastically and therefore increase QUIC's traffic share.

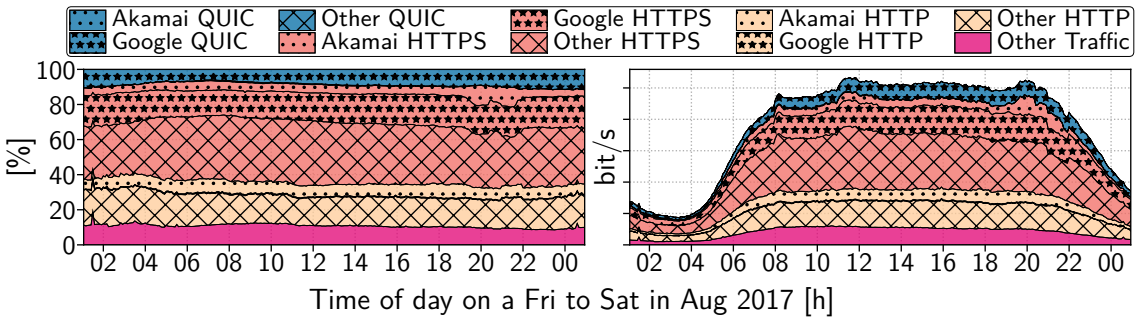


Figure 3.28 Mobile network traffic share of QUIC in a major European ISP. Left, relative share of QUIC traffic. Right, absolute traffic share compared to HTTP(S), y-axis has been anonymized at the request of the ISP.

In contrast, Akamai only serves 0.1% of its traffic via QUIC — despite contributing a large portion of the overall QUIC-capable IP addresses (40.71%, see Section 3.2.4.1). This discrepancy between the number of IP addresses and the traffic share suggests that QUIC is not yet widely activated among all customers/products. Still, on average, Akamai accounts for 10.3% (HTTP) and 5.1% (HTTPS) of all traffic, and thus, together with the fact that they already have a QUIC-capable infrastructure, has the potential to shift more traffic towards QUIC. A higher QUIC share has several implications, while QUIC and TCP are generally similar, subtle differences in the protocols may influence the performance of whole networks, e.g., by default QUIC uses larger IWs than those *standardized* for TCP by the IETF and demands to pace for smoothing the traffic. Further, network operators have raised serious concerns regarding the manageability of their networks [IET17a]. As QUIC fully encrypts all headers and thus hides signaling information from a passive observer (which generally enhances privacy), network operators cannot use passive network traces to estimate KPIs such as the RTT[20], reordering, or loss.

Takeaway. *As we have seen, Google already pushes 39.1% of its traffic via QUIC, thus network operators may already be challenged in parts of their network. An Internet giant, such as Google with its own browser, server infrastructure, and services, has the power to transform within short amounts of time regardless of operator concerns.*

Mobile ISP. The ISP supplied us with information which traffic is for their mobile (cellular) customers, which we show in Figure 3.28. Please note that Figure 3.27 also contains this reported mobile traffic. In contrast to the entire network of the ISP, the mobile traffic shows a different traffic pattern: while the throughput also decreases overnight, mobile traffic rapidly increases in the morning and stays rather constant over the day.[21] Apart from this, the average QUIC share in the mobile network of 9.1% (σ: 1.4%) is the highest share among all traces (see Table 3.9). In contrast, among the entire mobile Google traffic, only 34.0% (σ: 2.6%) is served via QUIC, lower than overall for the ISP. Also for mobile traffic, Akamai only serves a

[20] iQUIC, after lengthy discussions in the IETF offers a SPIN-bit that can be used to passively observe the RTT, see [DBK+18] for an in-depth analysis and explanation.

[21] We also verified with the ISP that the links are not fully-loaded and more capacity would be available.

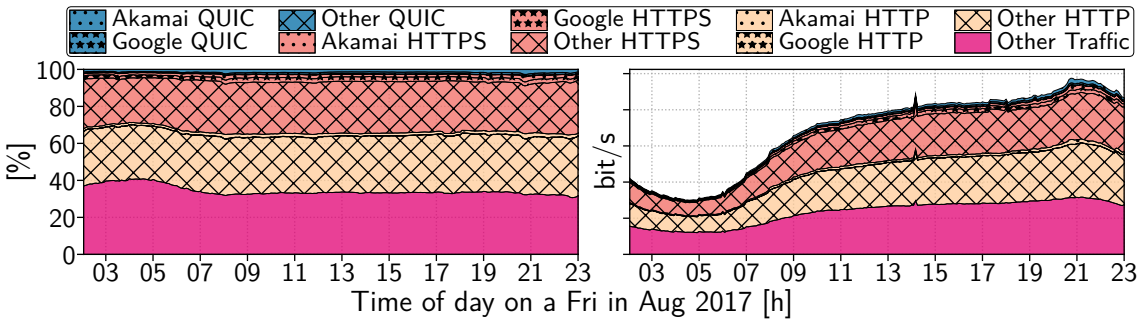


Figure 3.29 QUIC traffic share at a large European IXP. Left, relative share of QUIC traffic. Right, absolute traffic share compared to HTTP(S), y-axis has been anonymized at the request of the IXP.

negligible share of its traffic via QUIC and thus has the potential to increase the QUIC traffic share.

Takeaway. *QUIC traffic shares do not (yet) reflect server support. While Akamai operates a comparably large infrastructure in the number of QUIC-capable IP addresses, QUIC traffic is (still) almost entirely served by Google: this is likely to change.*

European IXP. We obtained sampled flow data of a large (European) IXP for the same day in August as for the ISP, and we show its traffic share in Figure 3.29. We classify Google and Akamai traffic by customer port information — since both peer at the IXP — and plot their HTTP(S) and QUIC traffic shares similar to the ISP. On average, QUIC accounts for 2.6% (σ: 1.0%) of the traffic, which is the lowest share among all traces (see Table 3.9). Unlike the ISP, Akamai now supplies the largest portion of traffic (59.9%), and we observe a lower share of Google traffic (33.1%) — recall that Google contributed 98.1% of the QUIC traffic at the ISP.

Discussion. We observe different QUIC traffic shares at the ISP/IXP and particularly different shares of the QUIC traffic by Google and Akamai (relative to the overall traffic of each vantage point). Different traffic engineering (TE) strategies likely cause these vantage-point-dependent differences, since both providers peer at both vantage points. These differences highlight that observed traffic shares are, in general, highly vantage-point-dependent. Understanding the incentives for these different TE strategies is an interesting starting point for future research.

3.2.5.2 Beyond the Census: Traffic Shares in Access Networks

Since our census in October 2017, we have observed significant increases in gQUIC-capable infrastructure (see Section 3.2.4.2), it opens the question if QUIC *traffic* also increased. We therefore now look at QUIC traffic through the lens of eyeball networks.

We were able to obtain six additional traces from the ISP: *i)* one day in the mid of March 2018, i.e., when Akamai did not yet reactivate their infrastructure, *ii)* one full week from the 30th of March to the 5th of April, 2018, i.e., during the week that Akamai's infrastructure increased, *iii)* the 20th of March and the 4th of May, 2018,

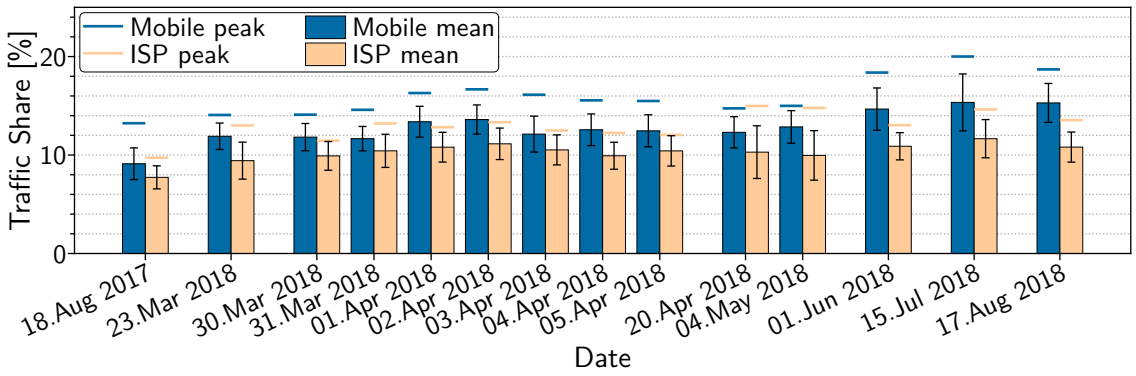


Figure 3.30 QUIC shares over all ISP traces, mobile network shown separately. Error bars denote the std. dev. And the horizontal lines denote the respective peak share.

i.e., two weeks and one month after the infrastructure activation, *iv)* the 1th of June, 2018, i.e., when Akamai announced that they would serve QUIC, *v)* the 15th of July, 2018, i.e., over one month after Akamai claiming to have it activated, and *vi)* finally, precisely on the same weekday in August one year after our initial trace that we inspected in Section 3.2.5.1.

Given that there is gQUIC support in Chrome, and currently no iQUIC support in any production browser, the traffic in this timeframe likely stems from gQUIC[22]. For our first analysis, we focus on the overall QUIC share and peak QUIC shares across all traces.

QUIC Traffic Shares

We again investigate QUIC traffic shares in the ISP and in its mobile network. We show the evolution of QUIC traffic in this ISP by its mean (bars) and the standard deviation (errors on the bars), and peak (horizontal lines) traffic share for each day and show it in Figure 3.30.

For all dates, we find that the mobile network contains a higher share of QUIC traffic compared to the whole ISP. These shares increase throughout our observations. The first to last observation point spans one year, within these extrema, we observe that the average QUIC traffic share has increased from 9.1% (7.8%) to 15.3% (10.8%) in the mobile network (whole ISP). Similarly, the peaks for the mobile network (whole ISP) have increased from 13.2% (9.7%) to 18.7% (13.5%), yet we find the highest peak in mid-July (mid-April) of 20.0% (15.0%).

Since network traffic is known to be dependent on the weekday, we investigate one full week starting on Friday the 30th of March until Thursday the 5th of April. In this week, we observe the highest QUIC traffic shares on Monday and Sunday, which then decrease during the week. Still, on average, the shares are very comparable since their average fluctuations are within the typical daily variations. Throughout that week, all shares (even the peak shares) are within 2.5% points of each other.

[22]The measurement inaccuracies highlighted in Section 3.2.5 still apply.

Takeaway. *The Internet transports an increasing amount of its traffic via QUIC. Our ISP's mobile network shows a significantly larger QUIC share than the rest of the network. We observed peaks up to 20.0%.*

While this analysis provides an overview of the overall traffic share, it fails to illuminate which networks produce this traffic and how much traffic in these networks is already QUIC which is the focus of the next section.

QUIC Shares by Operator

In Section 3.2.4.2, we have seen that Akamai has significantly increased their infrastructure support at the end of March 2018 after shutting it down in late 2017. Further, Akamai has announced [AC18] to increase the QUIC share for a part of their services starting on the 1st of June, 2018.

Previously, as of August 2017, we found Akamai to contribute only an insignificant portion of all QUIC traffic (0.1%) within our ISP. Please note, that we do no expect Akamai's traffic share to scale with their number of IP addresses, since, as a CDN, only a fraction of all their IP addresses will handle our ISP's traffic. Still, we did find that Akamai handles a significant amount of all traffic, i.e., potential QUIC traffic.

Since we found that the mobile network contains most QUIC traffic, our first analysis focuses on its evolution. Figure 3.31 and Figure 3.32a together show four full days in the mobile network (other days are similar). Given our observations in Section 3.2.4.2, we differentiate between Google (stars), Akamai (dots), and the rest (crosses). We further highlight the traffic mix between HTTP (yellow), HTTPS (red), and QUIC (blue).

Mobile Network. After one year, we find that our initial observation from 2017 did not change much. We still find that Google contributes nearly all QUIC traffic. Thus also the increases that we observed can be attributed to Google.

This increase can have multiple reasons, for once, Google's total traffic share has slightly increased, more customers utilize a Google product that is QUIC-enabled (e.g., Chrome), or Google's QUIC shows a better performance than TCP and Google thus chooses it for more users. Supporting this, we observe that over the year, Google delivered, on average (in chronological order of the plots), 64.4%, 45.9%, 44.7%, 45.0% of their traffic on average via HTTPS. On the other hand, the average amount of QUIC traffic that flows from and to Google increased: 34.0%, 53.4%, 54.5%, 54.4%. Thus, more than half of the traffic between Google and our ISP's mobile network is now QUIC. For a mobile network operator, this means that most of the traffic cannot be directly used to measure the network quality for traffic flowing between Google and the ISP. Thus, an operator could choose to encapsulate all QUIC traffic, e.g., to be able to trace the order of packets traversing the network, which could lead to decreased performance when encapsulation consumes parts of the MTU[23].

Looking at daily changes, we observe that the pattern that we observed in 2017 did not change, with one exception on the 15th of July with two peaks between 16:00 h

[23]Google's QUIC code defaults to a conservative maximum packet size of 1350 byte, thus, currently, it is unlikely that encapsulation leads to further reduced performance.

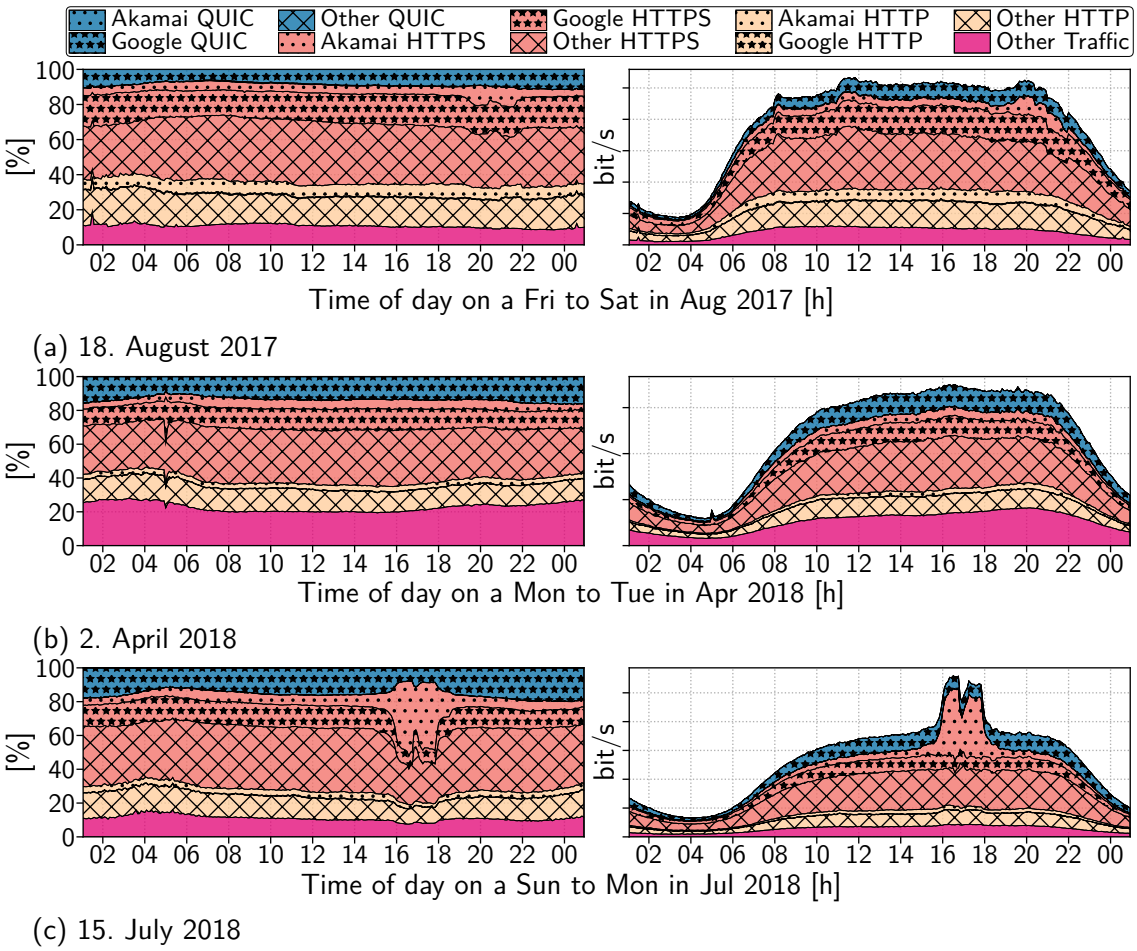
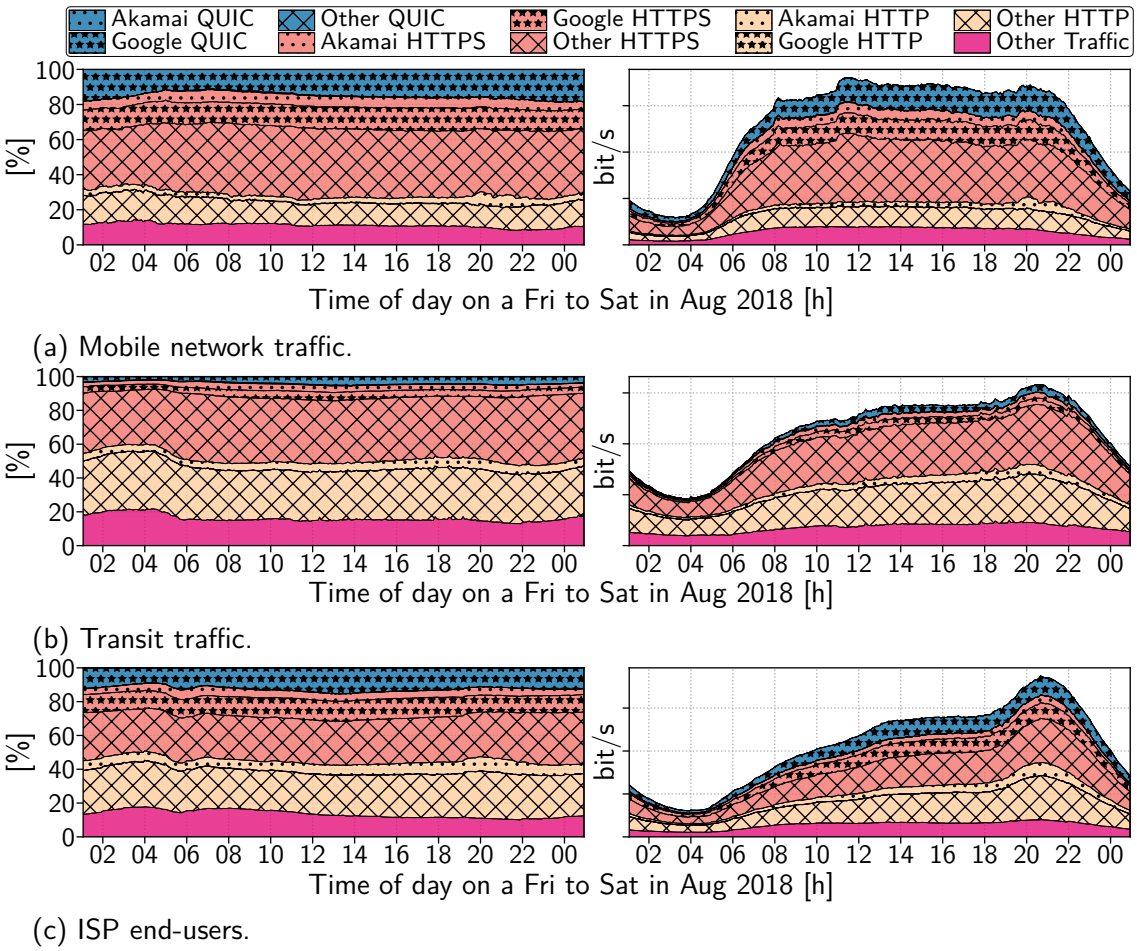


Figure 3.31 Mobile network QUIC traffic shares annotated by Google and Akamai. Left: Relative share. Right: Absolute share, please note that we anonymized the y-axis on request of the ISP.

and 18:00 h from Akamai via HTTPS[24]. This lead to a relative change in the QUIC volume, yet, the total traffic share stays similar.

Other Parts of the ISP. We continue our analysis by investigating how our ISP is influenced in other areas than the mobile network. Figure 3.32 extends our view by further investigating QUIC shares in transit traffic (Figure 3.32b) and towards non-mobile ISP end-users (Figure 3.32c). Focusing on Figure 3.32b, we observe that transit traffic of our Tier-1 ISP contains significantly less QUIC traffic than the other parts of the network. Since we found Google to be the only significant QUIC producer, this makes sense, Google and other CPs are known to heavily peer with eyeball networks having flattened the Internet hierarchy (see Section 2.1). We find 4.1% QUIC on average peaking at 5.7% on this day. When shifting to Figure 3.32c, i.e., to other ISP end-users, like DSL costumers, the relative distribution is similar to that of the mobile network. However, in comparison, we find slightly reduced amounts of QUIC: on average 12.8% peaking at 15.6% compared to 15.0% peaking at 18.7% in the mobile network.

[24]We believe both peaks correspond to the two halftimes of the FIFA World Cup final that was widely streamed via Akamai-based services.



(a) Mobile network traffic.

(b) Transit traffic.

(c) ISP end-users.

Figure 3.32 QUIC traffic split by source/destination annotated by Google and Akamai for the 18th August, 2018. Left: Relative share. Right: Absolute share, please note that we anonymized the y-axis on request of the ISP.

Akamai. In contrast to our observation (Section 3.2.4.2) that Akamai operates the most extensive QUIC infrastructure, we were surprised that they are not pushing large fractions of their data via QUIC, even after having announced to ramp up QUIC shares in June 2018. Still, we do find Akamai originating QUIC traffic in our datasets; however, compared to the total volume, it simply does not visually appear in our plots. For example, we find that Akamai contributed 0.3% of all QUIC traffic in our transit trace. We thus set out to investigate how Akamai's QUIC traffic changes for our datasets.

To this end, Figure 3.33 shows the evolution of QUIC for Akamai. To investigate how Akamai rolls out and tests QUIC, we plot the average share of QUIC compared to HTTP and HTTPS pushed by Akamai and normalize all observations to our initial observation from 2017. We see that for our second trace, QUIC traffic nearly ceases to exist which corresponds to our earlier findings that the infrastructure was shut down. We then see a gradual increase of QUIC when the infrastructure is reactivated, and especially the mobile network quickly surpasses the original share. Over the week in March/April, we see that the QUIC share fluctuates slightly. Since QUIC is likely not activated for all services, such variations could be explained by different

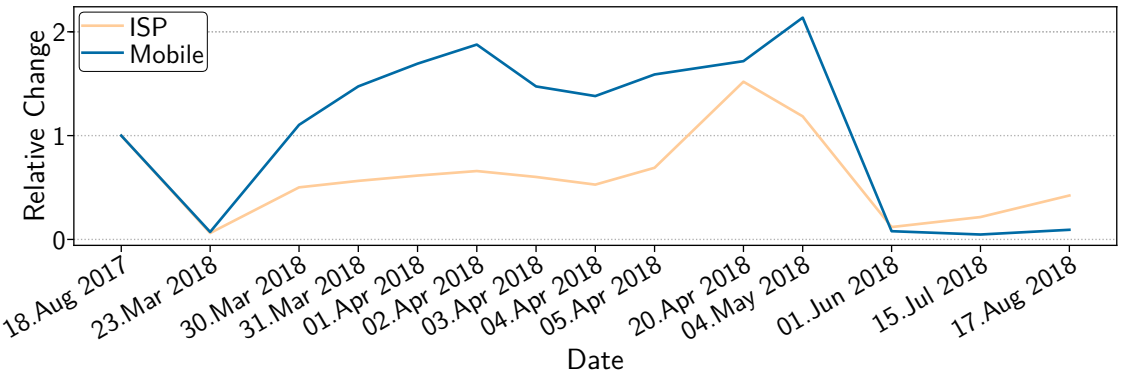


Figure 3.33 Change in QUIC share (from HTTP+HTTPS) pushed by Akamai, normalized to the first datapoint.

demand for different services. In June, i.e., at the officially announced Akamai QUIC enabling date, we find QUIC shares to go back to nearly zero. For the whole ISP, we observe that traffic is then again increased.

Takeaway. *We find QUIC in all parts of our ISP with varying traffic shares. While the absolute traffic peaks at a fifth of the total volume, traffic to individual peers can be dominated by QUIC (Google) demonstrating the need for manageability in QUIC. Even though Akamai announced a QUIC rollout, we still find Google to dominate the traffic mix, and it appears that Akamai tests QUIC on a small subset of their customers.*

As we have seen, QUIC is being largely deployed and mainly driven by Internet giants. While these giants seem to put a significant effort into making QUIC an Internet reality, our analyses up to now have not focused on whether or not it is actually beneficial for them. For example, Google acknowledges [LRW+17] that QUIC consumes roughly ×3.5 more central processing unit (CPU) time than TLS+TCP. This thus begs the question if QUIC is really so much better than TCP that these increased loads are worth the potential performance gains.

3.2.6 The Performance of gQUIC Against an Optimized TCP+ TLS+HTTP/2 Web Stack

Given QUIC's motivation to increase performance, it is no surprise that there are studies [CDM15, BG16, MKM16, CMT+17, KJC+17, YXY17] showing that QUIC clearly outperforms the traditional Web stack (i.e., HTTP over TLS over TCP). These studies are nevertheless subject to several limitations, e.g., many studies utilize the page load time (PLT) to measure performance. However, it has been shown [KRB+17, ZWH17, BDM+17] that PLT does not correlate well with *human-perceived* performance. Another shortcoming is that *all* studies compare a highly optimized QUIC against an off-the-shelf Web stack which, however, offers similar optimization potential, as we have shown in Section 3.1, that is not used. For example, Google QUIC shipping with the Chromium code utilizes packet pacing and an IW of 32 segments. In contrast, a stock Linux TCP stack with default settings

will not pace and defaults to an IW of ten segments, obviously disadvantaging the regular Web stack that can be avoided by parametrizing TCP similar to QUIC.

While QUIC offers a new level for protocol customization and evolution, it remains questionable from a performance point-of-view if switching to QUIC should be a top priority. To this end, we ask and answer three questions. First, does QUIC outperform a tuned TCP Web stack from a technical view? Second, do humans even notice a difference? Finally, if they notice a difference, how much of a difference does it make? That is, does QUIC actually impact the QoE?

To answer these questions, we first build a testbed that allows reproducible measurements of both protocols. This testbed enables us to perform an unbiased comparison between the two stacks. To this end, we are going to modify the Mahimahi [NSD+15] framework and incorporated the gQUIC Web stack. Our modifications grant us full control over the network parameters, i.e., bandwidth, delay, and queues as well as the client and server, i.e., enabling to modify the protocol's parameterization for comparable measurements. We then use video recordings of website visits in our testbed to perform studies in a controlled lab environment, using an online crowdsourcing marketplace, and a voluntary crowd study to investigate the QoE of QUIC.

3.2.6.1 Web Performance Metrics

We aim to evaluate the performance of the protocol stacks on a broad set of standard Web performance metrics. Besides network characteristics like goodput or link utilization as measured in [CDM15, YXY17], PLT is the most used metric. Nevertheless, PLT does not always match user-perceived performance [KRB+17, ZWH17, BDM+17], e.g., it includes the loading performance of *below-the-fold* content that is not displayed and thus not reflected in end-user perception. This lack of depicting user-perception is why we decide to focus more closely on state-of-the-art visual metrics that are known to better correlate with human perception. These metrics are derived from video recordings of the pages loading process *above-the-fold* as recommended by [BAM11, GDA17].

Metrics of interest are the time of the FVC, LVC, and time the website reaches visual completeness of a specific threshold in percent. In our case, Visual Completeness 85% (VC85), which corresponds to the point in time measured from the navigation start when the currently rendered website's above-the-fold matches to 85% the final website picture. Only navigation start can be used as a starting point since visual metrics are solely derived from video recordings. Lastly, we also take into account the SI [Goo19b].

3.2.6.2 Repeatable Protocol Performance Evaluations

To compare the performance of website loading processes subject to different protocol configuration, we require a controlled testbed approach that enables repeatability. Our testbed relies on a rich emulation framework that we build on top of the

Protocol	Description
TCP	Stock TCP (Linux): IW10, Cubic
TCP+	IW32, Pacing, Cubic, tuned buffers, no slow start after idle
TCP+BBR	TCP+, but with BBR as CC
QUIC	Stock gQUIC: IW32, Pacing, Cubic
QUIC+BBR	gQUIC, but with BBR as CC

Table 3.10 In total, we regard six different protocol configuration in our tests.

Mahimahi framework [NSD+15]. We will highlight the most significant changes as well as the parameterization that we apply for our performance measurements and the subsequent user studies. Furthermore, we rely on the Chromium browser to request websites as it has the most up-to-date and likely best-engineered support for gQUIC.

Enabling Repeatability. To enable repeatability in our testbed, we extend the Mahimahi framework [NSD+15] to replay websites. Mahimahi allows replicating the multi-server nature of today's websites by first recoding traffic which is subsequently analyzed to replicate the exact hosting structure. By default, Mahimahi does not support QUIC nor configuring transport protocol related parameters. To this end, we added these requirements by essentially replacing parts of the record and replay infrastructure of Mahimahi and incorporated the gQUIC Web stack in Version 43. We further replace the default Apache server by the NGINX Web server which is known to have better performance and is more familiar to us. We take special care in ensuring that the right protocol variant is chosen by explicitly enforcing Chromium to either deactivate QUIC or demanding it on specific domains.

Protocol Parametrization. For TCP, we configured pacing and an IW of 32 segments to match the gQUIC defaults. Moreover, we configured it to not fall back to the IW after idle, even though we expect our connections to not idle when requesting websites. Furthermore, we experiment with enabling BBR on both protocols. Table 3.10 summarizes the different protocol variants that we regard.

A single website visit in our testbed is done using a fresh Chromium browser with an empty cache. This starting from scratch has several implications on the protocols: it means that QUIC will *not* perform a 0-RTT connection establishment but a 1-RTT handshake. Since there is no support for TLS 1.3 early-data in Chromium (as of June 2019) and the limited and challenging deployment of TCP Fast Open in the Internet (see Figure 3.1 from our initial measurements in this chapter), this still gives a 1-RTT advantage of QUIC over TCP+TLS+HTTP/2 on paper. QUIC, similar to early-data and TCP Fast Open, suffers from replay attacks, which have an unusually large surface in distributed clusters [FG17] when requests are non-idempotent (see Section 2.3.2.1 for further details on the connection establishment and the problems with 0-RTT). To this end, e.g., Cloudflare only allows non-parameterized GET requests via TLS 1.3 0-RTT [Sul17]. In summary, there is currently a lack of signaling idempotency through the stack such that 0-RTT data processing could be enabled easily on a large scale. Thus, we believe that comparing a 1-RTT QUIC

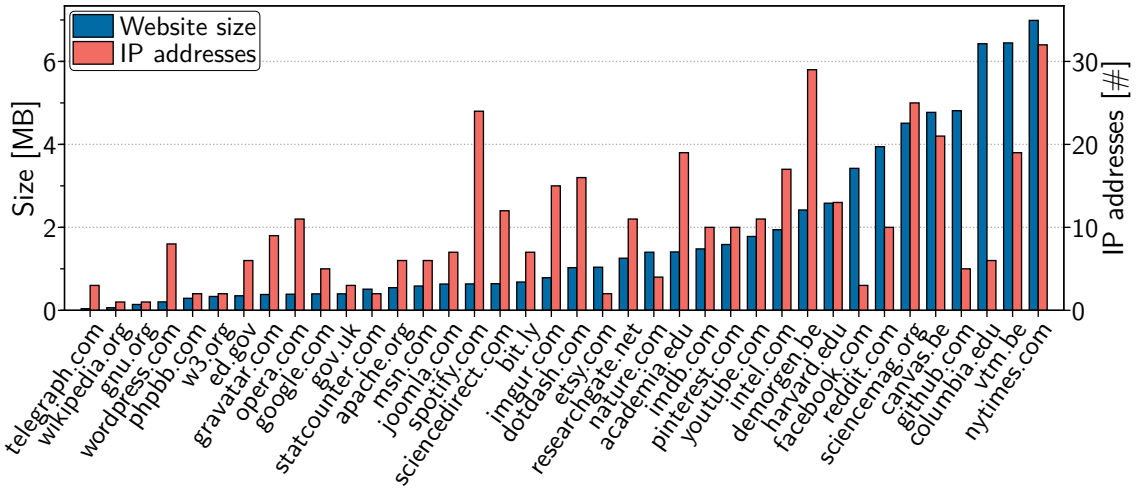


Figure 3.34 This figure depicts the download size of the replayed websites (blue) and the number of unique IP addresses that need to be contacted for resources (red).

with a 2-RTT TCP/TLS reflects a likely Internet-wide deployment of both, even though we will see 0-RTT QUIC and TLS 1.3 early-data in parts.

Websites Selection. Choosing websites that replicate a real-world picture of commonly used websites is challenging. Our goal is to obtain a small set of domains diverse in size, resources, and contacted servers. As there is no standard test set of such websites, we use the domain collection from [WMQ+18] consisting of 40 different websites from which we had to exclude two from the technical evaluation and two additional ones from the user studies. One domain is a private project website, and the other failed to record and reply properly. The two additional domains that we exclude from the user study take too long to load. The domains ultimately originate from the Alexa Top 50 and Moz Top 50 ranking lists and have been chosen in a way to obtain a good distribution of page size and resource counts [WMQ+18], see Figure 3.34. The bars in red suggest that the majority of our tested sites use a multi-server infrastructure, highlighting the relevance of replicating it with Mahimahi.

Network Parameter Selection. Within our study, we want to cover the performance in "good" networks as well as in "bad" networks. To this end, we select four different network settings, which we summarize in Table 3.11. We emulate the different network parameters with the help of the built-in tools from Mahimahi. We stack the following three network parameters from server to client with Mahimahi shells. First, we delay a packet in each direction, both adding up to the desired minimum RTT. Second, the link shell implements a drop-tail buffer limiting the throughput per direction. Finally, the loss shell drops packets at random for both directions equally. We configure it such that the chance for the successful transmission of two packets, e.g., request and response, equals 1-p, with p being the desired loss rate.

We chose the median bandwidth of German households according to the German Federal Network Agency's website [zaf18], which we refer to as DSL. This network has no artificial random loss, and we set a low minimum RTT to which the queue adds further jitter (up to 12 ms). Similarly, we use median bandwidth values for

Network	Uplink	Downlink	min. RTT	Loss
DSL	5 Mbit/s	25 Mbit/s	24 ms	0.0%
LTE	2.8 Mbit/s	10.5 Mbit/s	74 ms	0.0%
DA2GC	.468 Mbit/s	.468 Mbit/s	262 ms	3.3%
AMSS	1.89 Mbit/s	1.89 Mbit/s	760 ms	6.0%

Table 3.11 Network configurations. Queue size is set to 200 ms except for DSL having 12 ms.

German mobile users, which we refer to as Long Term Evolution (LTE). Even though it is a wireless link, we set no artificial loss as the link-layer would recover it. Still, the min. RTT is already higher. Furthermore, we allow up to 200 ms of queuing. Lastly, we take network parameters for two "bad" in-flight WiFi networks that connect either via LTE to the ground (Direct Air-to-Ground Communications (DA2GC)) or via an in-flight satellite connection (aeronautical mobile-satellite service (AMSS)). Those parameters have been established in [RNB+18] and are characterized by low bandwidths and high delays as well as high random loss (see Table 3.11).

Producing Videos. We perform the actual measurements inside a virtual machine equipped with six cores and 8 GB of memory running Arch Linux kernel Version 4.18.16. To measure a single setting, consisting of one website, network, and protocol configuration, we configure a Mahimahi replay shell with the desired network stack. A single setting gets measured over 31 runs to gain statistical significance and at the same time keep the number of runs/videos manageable. We utilize the Browsertime [sit19] framework to instrument the browser. It records videos of the loading process that we subsequently evaluate for the visual metrics. For each run, Browsertime opens up a fresh Chromium browser Version 70.0.3538.77. In total, this leads to 760 configurations (38 domains, four network, and five protocol settings). We validated that each run completed successfully by reviewing the video recordings manually. For the user study, we then select a single video for each setting that closely fits a typical recording by taking the video that is closest to the average PLT inspired by [ZWH17].

3.2.6.3 QUIC vs. TCP: According to Web Performance Metrics

We evaluate the performance difference with all metrics in the different network settings (across all tested websites) employing a performance gain. The following equation explains the calculation of the performance gain between a reference protocol, e.g., TCP and a protocol to compare with like QUIC. $\overline{X}$ correspond to the mean over the 31 runs.

$$performance\ gain_{QUIC}^{TCP} = \frac{\overline{X}_{QUIC} - \overline{X}_{TCP}}{\overline{X}_{TCP}}$$

If not stated otherwise, numbers provided in the text are mean performance gains across all websites for SI. Besides comparing means, we also utilize an analysis of variance (ANOVA) test to tell whether there is a statistically significant difference

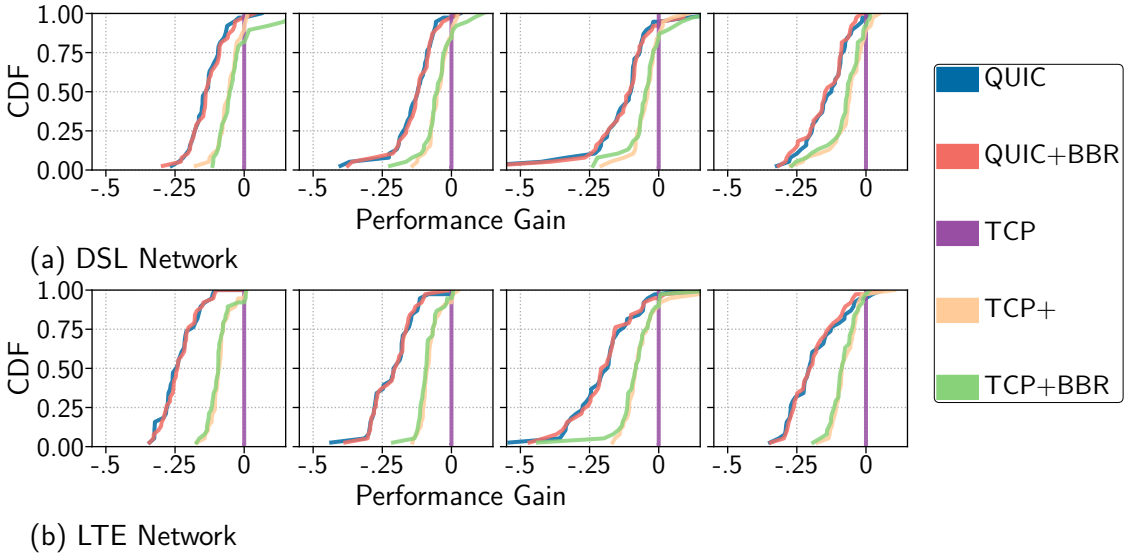


Figure 3.35 DSL and LTE network performance. CDF of the performance gain over all websites with TCP as reference protocol. If the performance gain is < 0 (left side of plot) then the compared protocol is faster than TCP.

in the distribution of the 31 runs of two protocols. If the ANOVA test for two settings is $p < 0.01$ (significance level), we count the setting with the lower mean as significantly faster; otherwise, we cannot conclude. The results of our measurements are depicted in Figure 3.35 and Figure 3.36. We show the cumulative distribution functions (CDFs) of the performance gain of the different metrics comparing stock TCP to the other protocol stacks. LVC is left out in these figures because, in contrast to PLT, there is no relevant difference visible.

DSL and LTE. For the lossless DSL and LTE scenarios, the protocols separate into two groups both yielding similar performance gains (see Figure 3.35). TCP+ (DSL: $-0.05^{\text{TCP+}}_{\text{TCP}}$, LTE: $-0.08^{\text{TCP+}}_{\text{TCP}}$) and TCP+BBR (DSL: $-0.05^{\text{TCP+BBR}}_{\text{TCP}}$, LTE: $-0.09^{\text{TCP+BBR}}_{\text{TCP}}$) perform almost indistinguishable but against TCP, there is a noticeable improvement visible throughout all metrics. Similarly, QUIC (DSL: $-0.09^{\text{QUIC}}_{\text{TCP+}}$, LTE: $-0.14^{\text{QUIC}}_{\text{TCP+}}$) and QUIC+BBR (DSL: $-0.09^{\text{QUIC+BBR}}_{\text{TCP+BBR}}$, LTE: $-0.13^{\text{QUIC+BBR}}_{\text{TCP+BBR}}$) perform equally but are still quite a bit faster than the two tuned TCP variants. For these two networks, the CC choice does not make a significant difference, which is likely due to the small queue. Stock TCP indeed lags behind all other protocols showing that stock TCP should not be used to compare against QUIC here. QUIC decreases the average SI by $-131.3\,\text{ms}^{\text{QUIC}}_{\text{TCP}}$ (DSL) and $-344.9\,\text{ms}^{\text{QUIC}}_{\text{TCP}}$ (LTE), but still also against TCP+ by $-87.1\,\text{ms}^{\text{QUIC}}_{\text{TCP+}}$ (DSL) and $-215.9\,\text{ms}^{\text{QUIC}}_{\text{TCP+}}$ (LTE).

In a second step, we take a look at the ANOVA test results focussing on DSL (LTE yields equivalent results). When comparing the runs of TCP+ and QUIC in DSL with PLT as the metric with each other, 30 of the 38 websites yield a significant improvement with QUIC. For the remaining eight websites, none was significantly faster than TCP+. For SI, even 32 websites are faster, and only six show no significant difference. Similar results can be seen when comparing QUIC+BBR with TCP+BBR this way. For TCP+ and TCP in the same scenario with PLT as the metric, 25 websites are faster with TCP+, for 12 websites, there is no significant difference,

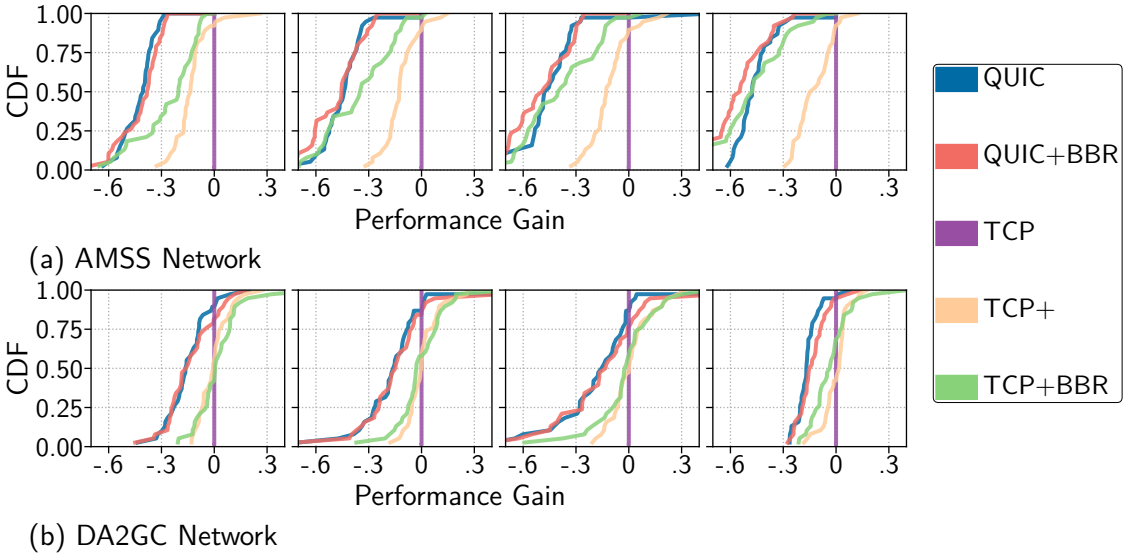


Figure 3.36 In-Flight WiFi network performance. CDF of the performance gain over all websites with TCP as reference protocol. If the performance gain is < 0 (left side of plot) then the compared protocol is faster than TCP.

and only one website was significantly slower. Again when comparing TCP+BBR with TCP+ and similarly QUIC+BBR with QUIC for DSL and LTE throughout all metrics, we find for the majority of the websites no difference. These results line up with the results shown in Figure 3.35. Moreover, the steep incline of the CDFs for QUIC and TCP+ indicate that the website size or structure seems to have little influence on the achievable gain. Only looking at SI and VC85, we see a small percentage of measurements where QUIC has a significantly higher gain.

In-Flight WiFi. For the AMSS and DA2GC networks, the overall picture (see Figure 3.36) is quite similar — meaning QUIC as well as QUIC+BBR, are usually faster than TCP+ (AMSS: -0.36$^{\text{QUIC}}_{\text{TCP+}}$, DA2GC: -0.14$^{\text{QUIC}}_{\text{TCP+}}$) and TCP+BBR (AMSS: -0.18$^{\text{QUIC+BBR}}_{\text{TCP+BBR}}$, DA2GC: -0.10$^{\text{QUIC+BBR}}_{\text{TCP+BBR}}$). Still, there are some crucial differences. For the AMSS link with the highest loss rate (6%), TCP+BBR operates much better than TCP+ (-0.26$^{\text{TCP+BBR}}_{\text{TCP+}}$). Since BBR does not use loss as a congestion signal, it increases its rate regardless of this random loss. Thus, in this case, the choice in CC has a more significant impact on the performance than the protocol choice itself. At the time of the FVC, TCP+BBR is already -2866.2 ms (avg.) quicker than TCP+, but with each later metric, the gap widens so that at PLT, TCP+BBR can keep up the pace even against QUIC and is 11395.4 ms (0.21×) quicker. This overtaking highlights that TCP with BBR needs some time to catch up and thus affects the FVC much more than the later PLT. For the QUIC protocols, the picture is similar. At first, QUIC and QUIC+BBR are similarly fast and mostly better than TCP+BBR. However, as the loading process commences QUIC+BBR outperforms QUIC slightly, e.g., -1828.3 ms$^{\text{QUIC+BBR}}_{\text{QUIC}}$ better SI. QUIC with CUBIC, nevertheless, is reasonably fast being still a legit option to use. The shape of the performance gain CDFs of QUIC+BBR and TCP+BBR are very similar, especially for PLT, highlighting the influence of the CC once again. We believe that QUIC with CUBIC is still competitive due to QUIC's ability to circumvent head-of-line blocking and its

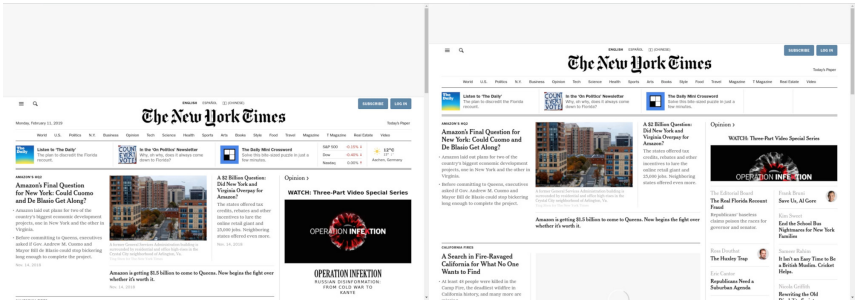


Figure 3.37 Screenshot during the loading process of the nytimes.com website. Left TCP right QUIC. QUIC in comparison delays a top banner leading to bad scores in visual metrics compared to the final website.

large selective ACK (SACK) ranges. For the AMSS network, QUIC reduces the SI by -8364.8 $\text{ms}^{\text{QUIC}}_{\text{TCP+}}$ (avg.) compared to TCP+ and by -2091.5 $\text{ms}^{\text{QUIC+BBR}}_{\text{TCP+BBR}}$ when taking both BBR protocols into account.

The last network, DA2GC, also has a high loss rate (3.3%) but much lower bandwidth. In this scenario, we observe no significant differences for most websites among all TCP configurations, even with the ANOVA test. We also see that in a small fraction of our measurements stock TCP outperforms QUIC and the tuned TCP variants. Nevertheless, again the QUIC variants are generally significantly faster with a higher performance gain at the FVC (e.g., $-0.14^{\text{QUIC}}_{\text{TCP+}}$) that persists towards the PLT (e.g., $-0.16^{\text{QUIC}}_{\text{TCP+}}$). The choice of the CC algorithm does not seem to have a significant impact here likely due to the low bandwidth. Only for PLT, we find QUIC with CUBIC to be slightly superior over QUIC with BBR. There is not a single website where QUIC+BBR yields significantly faster performance. The SI decreases with QUIC by -2632.5 $\text{ms}^{\text{QUIC}}_{\text{TCP+}}$ vs. TCP+ and by -1372.5 $\text{ms}^{\text{QUIC+BBR}}_{\text{TCP+BBR}}$ for BBR.

Metrics Discussion. Some of the websites exhibit miserable performance regarding the visual metrics VC85 and SI. We observe this behavior especially for the DA2GC network with performance gains of up to +1.0 compared to stock TCP (not shown, plots cropped for readability). The reason for these outliers is that the protocol choice has such a substantial impact on some websites that their resources load in different orders resulting in very distinct rendering sequences.

Figure 3.37 shows such a scenario exemplary for the nytimes.com website in the DA2GC network. Here TCP reaches VC85 after ∼48 s whereas QUIC needs ∼124 s even though the PLT for QUIC (∼141 s) is much faster than for TCP (∼170 s). For TCP the upper part of the website loads comparably early such that the lower elements are already rendered at their final positions. In contrast to that, QUIC manages to receive the lower contents first. Later, when the top banner completes loading, it shifts the whole website down. Therefore, VC85 fails to express this setting due to the shift. Similarly, SI is affected since it integrates over visual completeness over time. Thus, it critically depends on the website, the browser's loading order, and a user's preference for how a website should load to know which metric to use.

Protocol Design Impact. Within our testbed, any TCP configuration needs to fulfill two complete RTTs before the actual HTTP request can be sent out to

Net	Website	Metric	[ms]	[RTT]
DSL	gnu.org	FVC	0.5	0.020
DSL	wikipedia.org	FVC	-8.2	-0.341
DSL	gnu.org	PLT	1.6	0.066
DSL	wikipedia.org	PLT	-3.1	-0.128
LTE	gnu.org	FVC	0.6	0.008
LTE	wikipedia.org	FVC	-40	-0.538
LTE	gnu.org	PLT	-30	-0.412
LTE	wikipedia.org	PLT	-13	-0.175
AMSS	gnu.org	FVC	-196	-0.258
AMSS	wikipedia.org	FVC	-412	-0.542
AMSS	gnu.org	PLT	-1100	-1.447
AMSS	wikipedia.org	PLT	-529	-0.696
DA2GC	gnu.org	FVC	-130	-0.497
DA2GC	wikipedia.org	FVC	-1384	-5.283
DA2GC	gnu.org	PLT	39	0.150
DA2GC	wikipedia.org	PLT	-1005	-3.834
AMSS	gnu.org	FVC	-404	-0.532
AMSS	wikipedia.org	FVC	-143	-0.189
AMSS	gnu.org	PLT	-477	-0.628
AMSS	wikipedia.org	PLT	451	0.593

Table 3.12 Difference between the means over the 31 runs of QUIC and TCP+ when subtracting one RTT. Values < 0 denote that QUIC was faster. The lower AMSS table compares QUIC+BBR and TCP+BBR.

the server — TCP handshake plus TLS setup. In contrast, gQUIC requires only one RTT to do so — the first CHLO gets rejected by the server since the server certificates are unknown to the client. We are interested in whether this one RTT difference can explain the remaining performance gap between QUIC and TCP+. However, the complex interactions with multiple servers complicate an analysis since these connections are interleaved; simply subtracting one RTT is not possible. We, therefore, take a look at two websites served only via a single IP address (see Figure 3.34): wikipedia.org and gnu.org. We subtract one RTT from the FVC, as the earliest metric and one RTT from the PLT as the latest completing metric. Table 3.12 shows the results in the different network settings for TCP+ and QUIC and additionally for AMSS using the BBR variants of both.

For DSL and LTE, the corrected difference is below one RTT, and there are even three cases where TCP+ is slightly faster now. For AMSS in all cases with CUBIC as the CC, QUIC is faster but only to a maximum of 1.4× RTT. Since CC has a huge impact within this network, we also consider BBR here. Overall in AMSS with BBR, the difference is also below of one RTT, and for wikipedia.org and PLT even TCP+BBR is faster. Instead with DA2GC, the outcome is clearly for QUIC for the wikipedia.org website. Table 3.12 nicely shows that QUIC's RTT-reducing design clearly improves the performance. Even though, TCP Fast Open and TLS

1.3 early-data would close the gap, especially Fast Open remains challenging to deploy. Furthermore, having no head-of-line (HoL) blocking could still be a reason why, in the majority of the cases, QUIC is still slightly faster, especially when the networks are lossy. We expect further improvements when using 0-RTT connection establishment with QUIC.

Takeaway. *Our technical evaluation showed that tuning TCP parameters has a tremendous impact on the results for performance comparisons which can not be neglected when comparing TCP and QUIC. Still, in many settings, QUIC's performance is superior, but the gap narrows. Moreover, we find that QUIC's higher performance is caused mostly due to its superior design during the connection establishment. We assume that besides the RTT-reducing design, features like no head-of-line blocking increase QUIC's performance, especially in lossy networks. In those lossy networks, we also find that the choice of the CC algorithm has a much more significant impact than the protocol itself.*

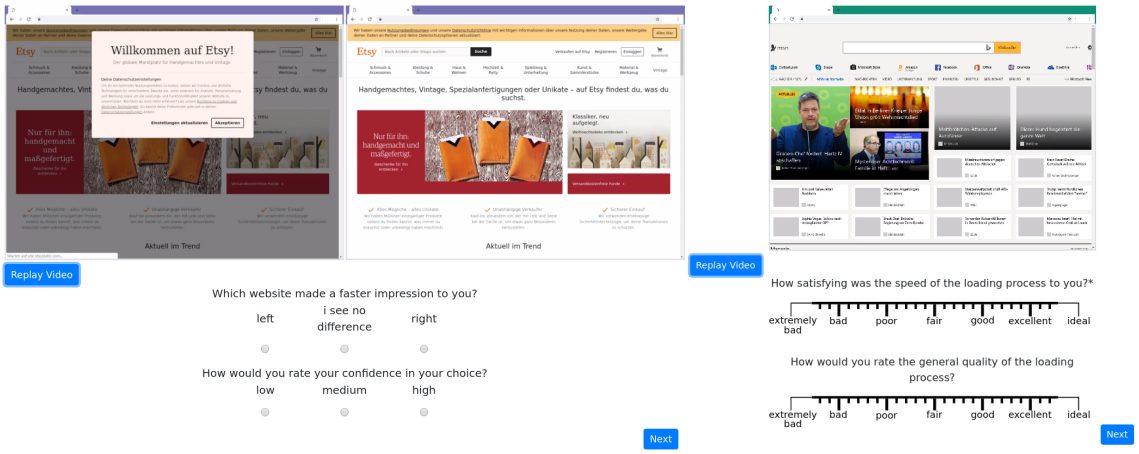
Still, even though the technical metrics were chosen to reflect human perception, it is unclear how well they really match actual human perception as this has not been explored for QUIC yet.

3.2.6.4 QUIC vs. TCP: According to User Perception

We complement our technical evaluation of QUIC using two user studies. Even though we found that QUIC outperforms TCP and even a tuned TCP in many setting when we compare human-centered metrics, this cannot answer whether or not these differences are actually above any perceptual threshold. The key focus of our studies is thus to evaluate the effect of protocol performance on user perception.

Study 1 (A/B): Do Users Notice? We begin by performing a *just noticeable difference* test to identify if users notice a protocol switch. The study design involves a pairwise comparison where two recordings of the loading processes of the same website with different protocol configurations but the same network setting are shown side-by-side (rendered into a single video) to participants (see Figure 3.38a). This pairwise comparison allows us to detect even subtle differences in the loading processes. After watching the video, the participants are asked to answer if the left or right video was the faster one or if they cannot decide. We furthermore ask them to rate their confidence in their choice.

Study 2 (Rating): Do Users Care? While the first study informs us if protocol switches lead to perceivable differences, it does not tell how users rate the perceived quality of the loading process. We answer this aspect in a second study, in which we only present one video (see Figure 3.38b) to let participants rate *i)* their satisfaction with the loading speed and *ii)* the general quality of the loading process. Both ratings are obtained on a seven-point linear scale [ITU03] ranging from extremely bad over bad, poor, fair, good, excellent to ideal, mapped to values from 10 to 70 with equidistance selectable by participants with a granularity of 1. To set a context for assessing speed perception, we ask the participants to consider being in a particular environment for this study: imaging being *i)* at work, *ii)* in their free time, or *iii)* on a plane.



(a) A/B study with video and questions.

(b) Rating study with video and questions.

Figure 3.38 Example screenshots of the two user studies. Note that usually the questions are hidden and pop up one after the other.

We implemented the user studies using TheFragebogen [GOR+19] and host it on our own infrastructure. Each study begins with a tutorial that explains the purpose and the procedure of the study. By informing the participants on the study goals and its procedure, it also aims at reducing noise in the responses.

Pilot Study. We tested our study in a pilot study before releasing it. It involves volunteers (friends and colleagues) testing our system to see if people that are unfamiliar with the study can perform it. The results of the pilot study are not used for evaluation, and participants did not participate in both the pilot and the later study to limit bias by training effects. The main feedback was that people were overwhelmed when the videos start without a tutorial up front, which we added. Secondly, we also rendered a Web browser window around each video (also shown in the figures) as we got the feedback that otherwise, people were unsure about the bounds of each website.

Performing the User Studies

We utilize three different subject groups for our user study.

Lab Study. First, we perform a controlled lab study with both the rating and the A/B study (the beginning is randomized), where we can monitor and supervise the study. Since the lab supervisor monitors the user behavior during the lab study (e.g., to check that participants properly conduct the test, i.e., actually watching the videos and not clicking randomly on the scales), it serves as our control data to evaluate the other two uncontrolled crowd-sourced groups. As this control group is rather small, we only consider five domains (wikipedia.org, gov.uk, etsy.com, demorgen.be, nytimes.com), which are diverse in website size such that the overall duration for each participant is roughly 10 min. This constraint leads us to show 28 videos for the A/B study and 11 in the work, 11 in the free time and only five, due to the increased video length, in the plane environment for the rate study.

Crowdsourcing Studies. We employ crowdsourcing to enlist a more extensive number of participants. As a second group, we recruit paid Microworkers [Mic19] participants. We follow the platform's guidance and offer 0.75 USD for a study between 10 min and 15 min length and allow a single user to only participate in each study once. After a Microworker (µWorker) completed the study, she can redeem her payout using a code that the study displays at the end. We show 26 videos in the A/B study and again 27 (11 work, 11 free time, and five plane) in the rating study. Third, we advertise our studies on social media to recruit regular Internet users. Since we expect unwillingness to perform a lengthy study, we show only 14 videos in the A/B and 15 videos (six work, six free time, three plane) in the rating study.

Conformance Filtering. While the controlled lab study helps us in judging the quality of the crowd-sourced results, we take extensive measures to ensure valid results since we suspect that at least on the Microworker platform people will cheat the system to solve the study quickly. To this end, we implement seven rules that are used to filter invalid results:

- **R1:** A video in the study has not been played.
- **R2:** A video has stalled.
- **R3:** There is a focus loss event (e.g., website not the active tab or window not in the foreground) for longer than 10 s during the study.
- **R4:** A vote was placed before the FVC.
- **R5:** A study took longer than 25 min, or a question took longer than 2 min.
- **R6:** A randomly placed control video was answered wrong. In the A/B study, we embed significantly delayed variants of the left or right video or have the same video on both sides[25]. In the rating study, we embed a very quickly rendering website and a very slow one; we expect the rating to be at least ten points apart.
- **R7:** A control question was answered wrong. Every fifth video includes an additional question that asks for the color of the browser frame, which is still visible while answering the question (see, e.g., Figure 3.38b having a green browser frame). Each video is assigned a random color from red, green, and blue; we chose the exact colors to be colorblind safe.

Table 3.13 summarizes the participation and how many results we removed due to each of the filters. Since we allow each µWorker to only participate once[26] in each study, and we suspect Internet users not to repeat the studies, these numbers should be close to the true number of individuals participating. Focus loss (R3) and voting before the FVC (R4) filtered the most results.

[25]Since even in the lab study people claimed to see a difference, we allowed left or right as a valid answer if the confidence was low.

[26]We cannot filters users with multiple µWorker accounts.

		-	R1	R2	R3	R4	R5	R6	R7
Lab	A/B	-	-	-	-	-	-	-	35
	Rating	-	-	-	-	-	-	-	35
µWorker	A/B	487	471	441	355	268	268	239	233
	Rating	1563	1494	1321	1034	733	723	661	614
Internet	A/B	218	217	210	196	171	170	159	155
	Rating	209	204	194	172	152	151	140	138

Table 3.13 Participation in our studies and results after each filter rule, final participations are underlined.

Ethical Considerations. Our study design follows standard guidelines for conducting crowdsourcing QoE studies [HKH+14]. Each participation (in the lab, from the Internet or as paid workers) takes place voluntarily. For the Microworker platform, we follow the platform's guidelines for payment. We chose not to pay lab participants to allow participation without monetary pressure. The user studies clearly state which data we are gathering, and only after completion of the study, this data is uploaded securely to our servers. Regarding the stimulus, the content in all videos does not show any sensitive material, e.g., violence, abuse, or other questionable content. In case of difficulties or errors, we are reachable via email on the study website and directly via the Microworker platform.

Study Agreement

We first compare our controlled lab study against both crowd-sourced studies. For the A/B study, on average, lab participants took 17.69 s, µWorker 14.46 s, and Internet user 15.59 s per video. The rating study took a little more time, lab participants took 21.44 s, µWorker 17.71 s, and Internet users 19.23 s per video. We found lab participants replay videos more often, especially in the A/B study. Regardless of group, faster networks resulted in more replays, which might already indicate that it is harder to spot differences. Regarding demographics, 76% to 79% were male. Within the Internet and the Lab group, the majority was younger than 24 years, for the µWorkers two-thirds were between the age of 25 to 44 years.

Figure 3.39 shows the agreement of all three groups in the rating study across all conditions on the x-axis (conditions in the rating study are single video and video pairs in the A/B study). The lab, as well as the µWorker data, is normally distributed. Thus, we show the mean and the 99% confidence interval (shared area) of the votes. We find that the µWorkers seem to fall mostly within the confidence intervals of the lab study, and hence, we believe that these votes are legit. In contrast, for Internet votes, we are unable to estimate the distribution and thus show the median of the votes. As is visible from the figure, the Internet group deviates most from the other two, and the number of votes that we were able to collect is lower. Consequently, we exclude it from further discussions, highlighting the challenge when trying to collect voluntary user data while setting a high standard on compliance with basic rules.

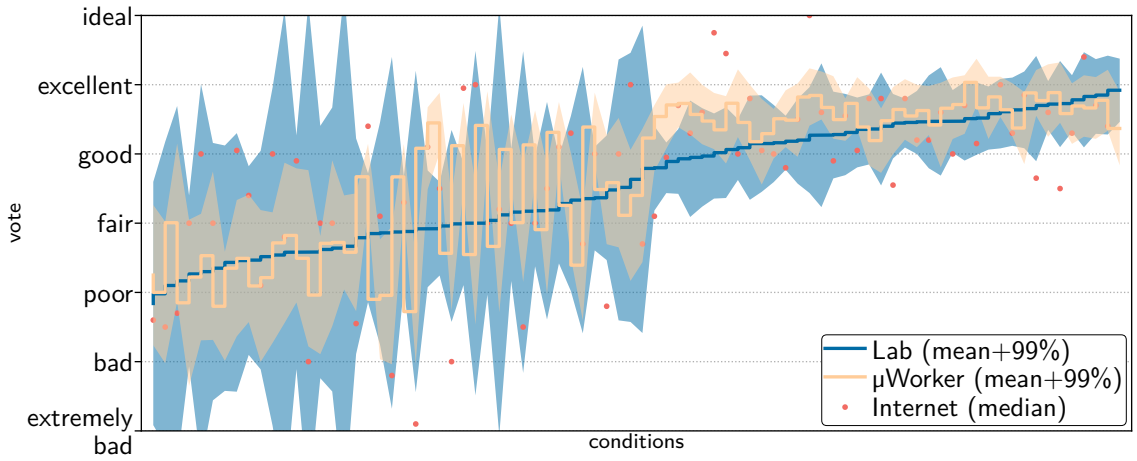


Figure 3.39 Rating study votes over all lab-tested conditions (ordered by mean vote of the lab participants). We show the 99% confidence interval. Internet values are not normally distributed, and thus, we show the median.

For the A/B study, in general, the agreement follows a similar scheme (not shown), but outliers look more severe due to the 3-point-scale (left, right, no diff.). We manually inspect the significant outliers in both studies, and we found that the websites are structurally very different. Figure 3.38a shows such a case, after loading the actual content, a welcome banner pops up. Participants in the lab study made their decision after the banner loaded (see left video) while people in the crowd-sourced data seem to vote earlier according to when the actual website content is shown, and those decisions often do not agree across different protocol versions.

Do User Notice a Difference?

To answer the questions if users actually notice a difference between the different protocol variants, we look at the A/B study and compare their votes. Figure 3.40 shows the share of votes for preferring a specific protocol variant in the four different network settings across all websites. The colors denote different pairs of protocols under comparison, the hatches signal preference for one or the other. Furthermore, we display the average replay count as vertical lines for each group of comparisons. So, e.g., in the DSL setting, slightly over 25% prefer TCP+, over 60% see no difference, less than 10% prefer TCP, and on average, people replayed the video roughly 1.4 times.

In general, we observe that the agreement for observing a difference rises when the networks become slower. For example, in the DSL setting, for all but the QUIC vs. TCP comparison (red), most participants do not see a difference. The comparably high average replay count expresses the difficulty of spotting a difference in the DSL network. Still, in general, more people experience the protocol variant quicker that is supposed to be faster. Lowering the bandwidth towards the LTE setting, the majority of participants now clearly vote the supposedly better variant (this confidence is also backed by the lower replay counts). In the slower networks, there are slight differences. For DA2GC, TCP is now favored in contrast to our tuned

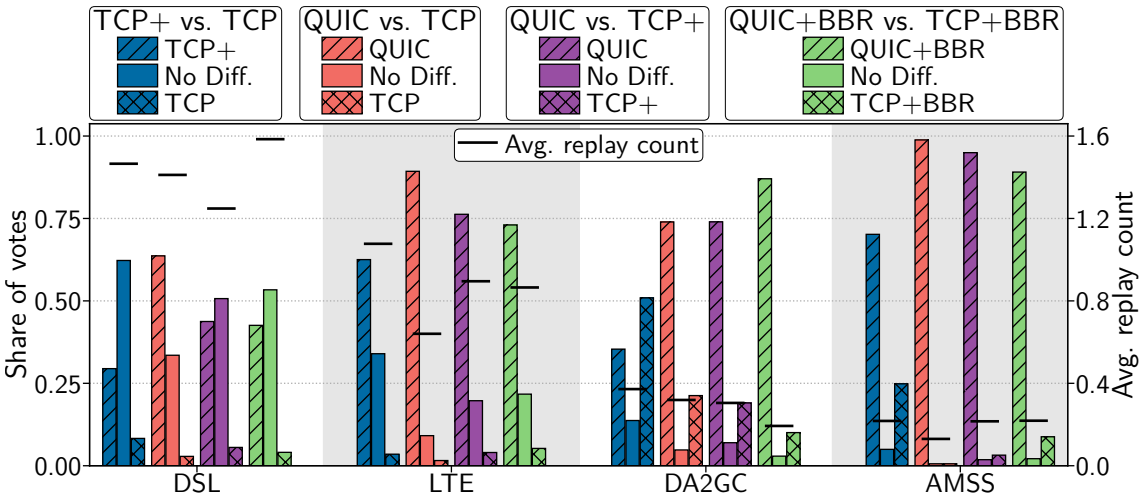


Figure 3.40 A/B study mean votes for each protocol combination depending on the network configuration.

variant (TCP+), we always found more retransmissions for TCP+ (on avg. ×1.5 but up to ×4.8) which may be explained by the comparably high initial congestion window leading to early losses. In contrast, QUIC seems to not suffer from the same problems (even though similarly configured to TCP+) as our participants experience it faster, we suspect that QUIC's large SACK ranges enable it to progress further and that the independent stream processing allows earlier renderings of the page. Looking at the AMSS network, the observation from DA2GC is now again reverted for TCP vs. TCP+; the increased bandwidth reassures our earlier assumption. For the other protocol variants, the picture from DA2GC is now even stronger with even more votes towards the supposedly faster variants. Again, the higher random loss-rate in this network backs our previous impression.

Takeaway. *In general, people do see a difference and indeed perceive QUIC as the faster protocol over TCP and even over a tuned TCP variant. However, in networks with high bandwidths, perceiving a difference seems to be more challenging.*

Do Users Even Care?

We continue to answer the second question, whether users care, or more specifically, we want to investigate if users perceive the already uncovered speedups as actually increasing the performance or if they cannot tell in isolation. To this end, we look at the results of the rating study, which we overview in Figure 3.41. In general, we observe that the work and free time scenario are rated similarly with a slight tendency towards better scores in the free time setting for DSL. In contrast is the plane setting (only having videos using the emulated in-flight networks) that shows only poor results.

When looking at the results within a network setting, we see only little variance between the different protocols, and the confidence intervals mostly overlap. When we test the different distributions for significance (using a significance level of 99% and an ANOVA test), we do not find any significant protocol/network configuration

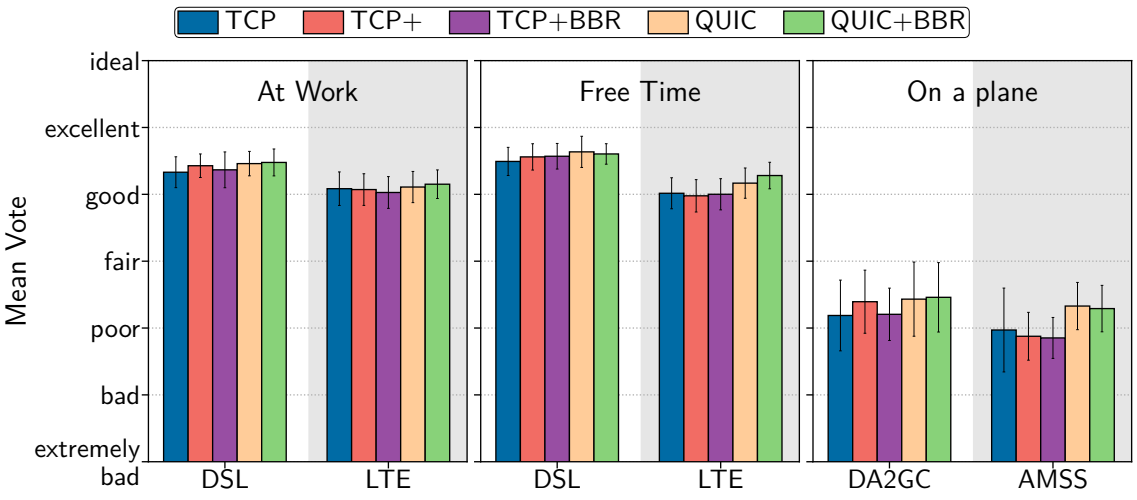


Figure 3.41 Rating study votes per protocol choice in the different settings. Error bars denote 99% confidence intervals.

that the users rate better. When we lower the confidence level to 90%, three settings are significantly different. First, in the LTE free time setting, QUIC+BBR is rated statistically more satisfying than TCP+BBR. BBR again makes the difference in the plane environment and the AMSS network. In the same setting, also QUIC without BBR is generally rated faster than TCP+. Thus, in general, there is little difference between the protocol variants. We now take a look at the specific websites where changes matter.

Where it Makes a Difference. In the DSL setting, eight websites show significant differences. Four of them rate QUIC faster than TCP, one faster than TCP+, and three rate QUIC+BBR faster than TCP+BBR. Spotify.com shows the largest difference with BBR. The website is small, but the browser has to contract many hosts. Still, we find small and also large websites that profit from QUIC.

In the LTE setting, only five pages show a significant difference. Our participants favor TCP+ over TCP once, otherwise QUIC over TCP+ (twice) and TCP (once). When using BBR, QUIC is favored once over TCP+. Regardless of CC, QUIC is roughly rated 10 points better, i.e., a whole quality level. Again, the websites show a wide variety of sizes and contacted hosts.

For DA2GC, we again find only five websites with a significant difference. Apache.org, a relatively small website in terms of size and resources, is preferred when delivered via QUIC in contrast to TCP and TCP+. When using BBR, google.com, gov.uk, and nature.com are perceived faster using QUIC. Lastly, w3.org is rated over 15 points faster when using QUIC in contrast to TCP+.

For AMSS, we find three pages. Wordpress.com is favored in the QUIC variant over both TCP variants, a website with few resources, small in size, and less than ten contacted hosts. Gravatar.com on TCP is liked less. Apache.org is favored when BBR is used.

Correlation to Technical Metrics. We next investigate which technical metrics (FVC, SI, VC85, LVC, and PLT) best reflect our participants' ratings. To do so, we

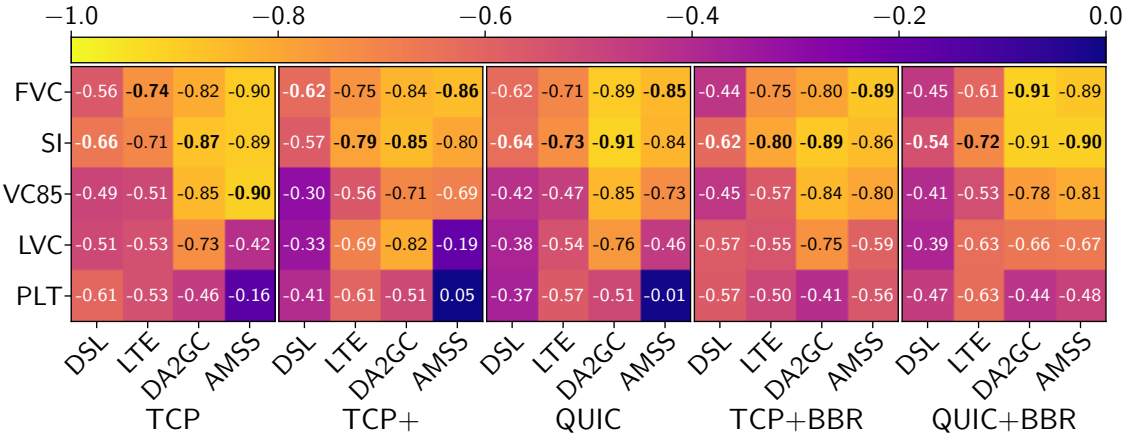


Figure 3.42 Pearson's correlation coefficient heatmap for different technical metrics to the user ratings in the different settings. High negative values (-1.0) are desired and mean that technical metric and user rating correlate perfectly. Largest correlation are in bold. For DSL/LTE, we chose the votes from free time setting.

calculate the Pearson's correlation coefficient of the votes compared to the technical metrics by first calculating the mean vote for each website and combining it with the technical metric. We chose Pearson's (e.g., over Spearman) because we are interested to see how well the linearity of the metric reflects the users' choices. Thus, we would assume a high negative coefficient when high vote scores are correlated with low metric scores (i.e., a quick loading) and vice versa.

Figure 3.42 shows a heatmap for the different correlation coefficients in the different network settings subject to the different protocols. As the figure shows, in general, SI shows the largest correlation even though we find that in speedy networks such as DSL, the correlation goes down and for slower networks, it goes up. However, this is not limited to SI but also holds for the other metrics, which seems to be in line with earlier findings that showed larger confidence in these networks. Opposing SI, we find PLT to have the worst correlation to our users' ratings, thus reinforcing related works.

Takeaway. *In general, our participants did not really care which protocol was chosen when given no direct comparison even though in some cases, QUIC showed a small tendency to be preferred. Furthermore, we found that the Speed Index shows the highest correlation with our participants' votes.*

3.2.7 Summary and Discussion

In this contribution, we have performed the first broad assessment of QUIC in IPv4 and various traffic traces and analyzed its potential to speed up the Web. That is, we study both the available *infrastructure* in terms of the number of gQUIC and iQUIC-capable IP addresses and domains and their *traffic share* at three vantage points. By probing the entire IPv4 address space, we find a growing gQUIC-enabled

infrastructure that is driven by Akamai and Google. While the infrastructure of iQUIC is tiny in comparison, we find that many new players have joined such as Facebook, Microsoft, or Cloudflare that further push QUIC. Still, we mainly find large Internet giants that experiment with QUIC. Driven by the fears of network operators that see QUIC as an end to passive measurability of the transport, we have analyzed and compared QUIC traffic shares from three vantage points, a campus uplink, an IXP, and an ISP. We have further extended our analysis to cover the evolution of QUIC in the ISP with 13 additional traces and find that QUIC traffic is growing and that Google already pushes more than half of its traffic via QUIC.

Further, we find traffic peaks of up to 20.0% in our sampled data demonstrating that QUIC has already transformed the Internet and has become a reality that operators have to face. Interestingly, this heavy push it not backed by QUIC's promise to rapidly speed up the Web. While our performance measurements showed that QUIC is often the faster protocol when looking at technical metrics, these wins diminish as users do not clearly perceive it as the faster protocol, especially when bringing TCP up to speed. Our studies further showed that CC has often a much more significant impact, but there are still instances where QUIC's advanced design also convinces real users, particularly when the networks are error-prone. Thus, it seems that QUIC may see clear advantages in countries where the network infrastructure is often based on wireless links such as in some developing countries. Still, in light of its current standardization, its foundation for HTTP/3 and the existence of multiple implementations by different vendors foreshadows an even larger QUIC-operated Internet once standardization finishes and QUIC is available as a default for browsers as well as Web server software.

This dominance demonstrates that Internet giants whose service is used by billions of people, and that control both client and server, have the power to transform the Internet without the consent of the networks that are involved. However, this remains a polarized topic: It takes Internet giants such as Google to develop and *test* a protocol such as QUIC on a large scale to be eligible to be picked up in standardization and to gain so much traction as we have witnessed in our measurements. This requirement to champion the protocol is especially apparent since most of QUIC's features have been around and were available in many preceding protocols, yet, none of these have made it to the Internet at large or seen deployment beyond academia. On the other hand, it is at the Internet giants' discretion to open up these developments and enable others to profit from their work, a business decision that should not be taken for granted and shows the blind spot in how Internet protocols can become a large-scale success. Luckily, Internet giants such as Google have done this in the past and their dedication to bringing their developments to the IETF for broader discussion, improvement, and reiteration currently enable an Internet-wide use of these protocols.

Still, we have also seen that these companies often work outside of recommended practice (as already highlighted in the previous contribution, see Section 3.1) and QUIC's realization in user space paves the way towards a rapidly evolving transport that can be updated as easily and as frequently as any application without requiring system reboots. Our measurements have already shown the rapid development of

QUIC and that new versions appear frequently. In light of these findings, we expect a highly dynamic future Internet transport landscape, and we especially point towards CC as a critical feature of each transport protocol. Our performance analyses already showed that in many cases, CC has a much more severe impact than the transport itself, and with QUIC, CC is now realized in user space. While this enables to modify CC easily, it is known that such modifications can quickly lead to unfairness as already witnessed for QUIC by Kakhki et al. [KJC+17]. The authors found that, at the time, QUIC's CC was unfair towards a single TCP connection, even though both were using CUBIC CC. It turns out that Google parameterized QUIC such that it emulated multiple TCP connections[27] to be competitive to a browser opening multiple TCP connections.

Our next contribution thus explores this largely neglected property of Internet transports, i.e., their usage and fairness in the Internet, outside of standardized and controlled lab experiments that will likely become more versatile in the future.

[27] A Google employee noted during a paper discussion at the ANRW 2019 that gQUIC's CUBIC implementation still emulates two New Reno flows. Video available at: `https://youtu.be/XNLi6KDYquU?t=3650`

3.3 Fairness in an Anarchic System – Congestion Control

The Internet has grown way beyond its original purpose of being a research network (see Section 2.1). Today, thousands of ASes connect and exchange data. The fundamental principles governing this data exchange are well established since decades and defined in the IETF's RFCs. To this end, the current best-effort Internet relies on CC to *i)* not collapse the network, and to *ii)* achieve fairness for flows competing for bandwidth at a bottleneck. For TCP, [RFC5681] requires the implementation of slow start, congestion avoidance, fast retransmit, and fast recovery (generally known as TCP Reno, see Section 2.3.3.1). Other algorithms improve on specific aspects of Reno, e.g., to enable higher performance over large BDP networks. Usually, when a new or modified CC algorithm is proposed, it is rated in terms of TCP fairness when competing with Reno or CUBIC as Linux's default CC algorithm. In 2005, Medina et al. [MAF05] showed that *most* Internet flows halve their cwnd on loss and are thus TCP conform, leading to an expected flow-rate fairness [RFC5290].

Since then, the Internet landscape has drastically changed, end-users use the Internet with increasing access speeds [Aka16], and content such as video is causing a substantial fraction of Internet traffic [TGD+18, EGR+11]. These increasing demands have led to a logical centralization of the content-serving Internet, where a few big players serve the majority of the content [CSA+18, LIM+10]. In our first contribution (see Section 3.1), we have shown that CDNs specialize in serving such content by tuning their TCP stacks beyond RFC-recommended values in hopes for higher performance and user satisfaction. Fundamentally, such observations raise the question of fairness, and in fact, from an economic standpoint being unfair to a competing CP might be advantageous (e.g., by being able to deliver data with more than a fair bandwidth share). While identifying a CP's CC algorithm (e.g., via [YLX+11, SRH+19]) helps in understanding its principal behavior or deployment, these works do not take into account the actual parameterization of the algorithms such as they are used in operation which has the potential of drastically changing the fairness. Moreover, transport protocol evolution with QUIC has the potential to further lower the hurdle for modification in the future, given its realization in user space for flexible customization (see Section 3.2).

In light of these historical changes, this contribution investigates the behavior and interaction of Internet giants and specifically large CPs. Thereby, we shine a light on current practices and evaluate the question of whether actual Internet traffic adheres to the conventional understanding of fairness (see Section 2.3.3 for an extended discussion on fairness). To this end, we devise a methodology that enables us to compare testbed results with actual Internet traffic. Specifically, this contribution extends the state-of-the-art by:

- Presenting a testbed methodology using RTT-fairness to study actual TCP traffic by major CPs to account for a broad set of TCP optimizations used in practice.
- Comparing the fairness properties of testbed hosts to actual traffic by six major CPs subject to different bandwidth, RTT, queue sizes, and queueing disciplines

in a home-user setting. We find that achieving a fair bandwidth share largely depends on the competing CC algorithms (CUBIC vs. BBR) and the involved network conditions.
- Finding that real Internet traffic has the potential of being *unfair* today but that individuals can reclaim control over how Internet giants use CC by deploying active queue management (AQMs) that guarantee fair and performant bandwidth allocations.

Structure. We introduce flow-rate fairness and related works in Section 3.3.1. We then introduce our testbed methodology and its validation in Section 3.3.2. Section 3.3.3 discusses the results of our fairness study before we conclude this contribution and this chapter.

3.3.1 Background and Related Work

One of the fundamental challenges in the Internet is the decentralized resource allocation of bandwidth (see Section 2.3.3 for a broader introduction to CC). However, TCP's initial design only prevented overloading single end-points and did not consider the possibility that the network itself could become overloaded and collapse upon this congestion. As centralized algorithms are not deployable on the Internet, decentralized CC was soon added to TCP's design. However, the highly distributed nature of the Internet quickly showed that there are scenarios where the early CC often yields less than optimal performance, which has led to a plethora of research for evolved and optimized algorithms. With the introduction of ever more algorithms, questions about their interaction arose, challenging how these algorithms share the available bandwidth. Research has hence also considered these aspects by investigating *fairness* of CC. Well-studied fairness measures are the intra-protocol flow-rate fairness, i.e., how well do two instances of the same algorithm share the available bandwidth, the RTT-fairness, i.e., what happens if the flows have different RTTs, and the inter-protocol fairness, where one investigates two instances of two different algorithms.

Intra-Protocol and RTT-Fairness. For CUBIC, research has commonly found decent intra-protocol fairness and an inverse-proportional RTT-fairness, meaning that instances with smaller RTTs get a larger share of the overall bandwidth (see Section 2.3.3 for more details). These findings have been confirmed for a broad set of different network characteristics, ranging from small (10 Mbit/s, e.g., [LSM07]) to large bottleneck bandwidths (10 Gbit/s, e.g., [XKC+14]) or short (16 ms, e.g., [HRX08, MBB08]) to long (324 ms, e.g., [HRX08, MBB08]) RTTs.

For BBR, less research exists, and the available studies partly disagree on the properties of BBR. This is especially true for intra-protocol fairness, Cardwell et al. [CCG+16a] claim a high degree of fairness across the board, while Hock et al. [HBZ17] identify scenarios where the fairness is significantly impaired. Regarding RTT-fairness, it is commonly found that BBR has a proportional RTT-fairness property, i.e., a flow with a larger RTT gets a larger share of the available bandwidth [CCG+16a, MJW+17, SJS+18]. Hock et al. [HBZ17] generally confirm the

findings but by investigating two different bottleneck queue sizes, they find that in scenarios with a smaller queue size (0.8× BDP) flows with a smaller RTT have a slight advantage, while in large buffer scenarios (8×BDP) the inverse is true and flows with larger RTTs have a significant advantage.

Inter-Protocol Fairness. While the intra-protocol and RTT-fairness of CC are essential for a massive scale-out of the algorithms, the inter-protocol fairness property shines a light on the co-existing use of different CC algorithms in the Internet. Unfortunately, several groups of researchers have found that BBR and CUBIC do not cooperate well, as CUBIC flows dominate BBR flows in scenarios with larger buffers (generally above 1×BDP) while the opposite is true for small buffer scenarios [CCG+16a, HBZ17, SJS+18].

Even though many studies investigate how specific algorithms affect each other, research misses an analysis which algorithms are genuinely used in the Internet. This attestation becomes even more challenging when considering QUIC, which hides the transport headers from a passive observer, a challenge that is actively researched [SRH+19]. Moreover, many studies neglect the parameterization and tuning potential of these algorithms that, e.g., Internet giants use in practice (see Section 3.1). We address this lack by exploring if actual Internet traffic of Internet giants — which carry the bulk of today's Internet traffic — still adheres to the conventional understanding of TCP fairness.

3.3.2 Methodology

CC research traditionally involves *simulation* or *testbed* studies, which give researchers *complete control* over the investigated scenarios. While this is desirable for controlled experiments, the involved abstractions and assumptions do not allow to cover real-world settings completely. For example, the employed algorithms and their parameterization in real-world systems are typically unknown. To study CC fairness in practice, we, therefore, contact real-world Internet systems with a testbed setup. This perspective enables us to still control *some* parameters (e.g., bottleneck bandwidth and delay) while studying the CC algorithms as run by real systems. This way, we can study if Internet traffic by CPs still adheres to the conventional understanding of TCP flow-rate fairness.

3.3.2.1 Home User (Residential Access) Scenarios

The fundamental design choice of our study is to investigate fairness from the perspective of an end-user accessing the Internet through a home router. Even though peering links have been identified as possible points of congestion [DCG+18], it is still widely believed that access links form the bottlenecks and thus congestion happens at the network edges, more specifically at the end-user's access link (in the downlink direction towards the user) [BCL09]. We model this scenario in the form of a simple dumbbell topology that is the foundation of our testbed, which we illustrate in Figure 3.43. The user — represented by the client — is connected to the testbed

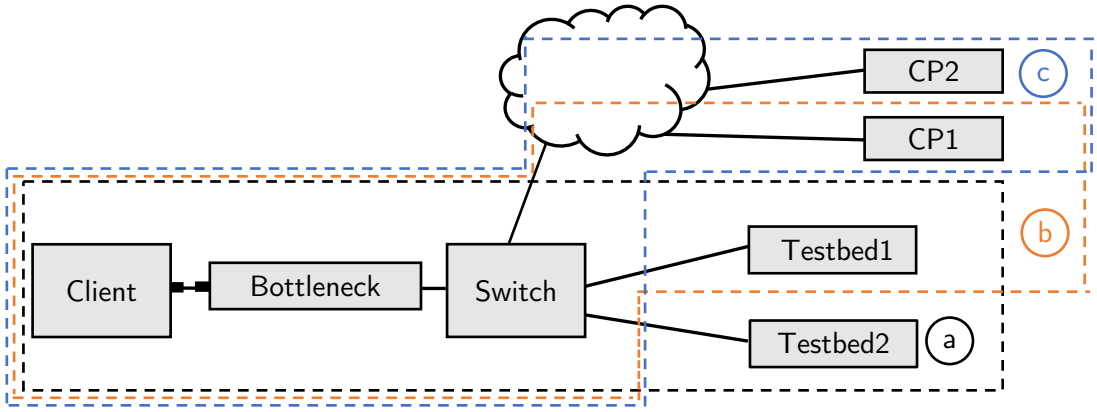


Figure 3.43 Testbed topology with testbed and online components. Scenario ⓐ (testbed-only), Scenario ⓑ (testbed & Internet), Scenario ⓒ (Internet-only).

network via a dedicated machine serving as a configurable bottleneck (via Linux's TC subsystem). In general, the client can request traffic from all kinds of sources, from within the testbed and from Internet sources. For our study, we focus on three distinct scenarios.

In Scenario ⓐ (testbed only), we investigate the out-of-box performance of CC by simultaneously requesting traffic from two testbed machines. This scenario, above all, establishes a baseline and identifies potential influencing factors on the overall interaction of CC. Building upon this baseline, Scenario ⓑ (testbed & Internet) then replaces one of the two testbed-flows with a flow originating from the Internet. Thus, we compare how the Internet flows interact with the out-of-box CC algorithms. Finally, Scenario ⓒ (Internet-only) considers the case where both flows originate from the Internet to investigate how and whether their interactions differ from the previous scenarios.

The common goal of all three scenarios is to judge the bandwidth sharing behavior of different CC algorithms in different network settings for which we consider four network characteristics. The bottleneck *bandwidth* and the overall *RTT*, as a result, give upper bounds (in terms of available data rate) and lower bounds (in terms of responsiveness) on the overall performance, while the bottleneck queue, characterized by its *queue size* and *queuing discipline*, introduces jitter, and loss.

3.3.2.2 Testbed Setup

Our testbed's core consists of one machine representing the end-user and hence serves as the client throughout the scenarios. Another machine which represents the user's access link and hence the overall bottleneck. The latter is then used to model all connection-specific properties like delay or bandwidth. For the scenarios where we create flows from within our network, we deploy one machine for each flow that is involved and configure server-side parameters like the deployed CC algorithm on them. All machines within the testbed use a Linux 4.13 kernel and are interconnected via Gigabit Ethernet to ensure that the physical links never become a bottleneck.

Limiting Bandwidth. Most configurations, like rate-limiting, are done on the bottleneck's egress queues. Here, we configure the bandwidth and queue size using a token bucket filter with a burst size of a single frame while using different queue management techniques (see Section 2.3.4 for a broader discussion of queuing mechanics). Even though Internet access links are often asymmetrical, we disregard this fact as we are not interested in investigating reverse-path congestion and use the same bandwidth in both directions.

Ensuring RTT-Fairness. To reason about the principal question of this contribution, i.e., about the bandwidth sharing properties of Internet flows, we employ the RTT-fairness property which states that two flows should share the bandwidth equally if they have the same RTT. This equality, in turn, means that we only consider those cases where the different flows have the same RTT. Consequently, we use fairness synonymously for RTT-fairness.

To add delay to our testbed, we use TC to perform ingress packet processing at our bottleneck. There, we redirect traffic to an intermediate queue disc enabling us to use NetEm to add a delay before we release the packet for forwarding to the actual egress queue. We do not configure any artificial jitter using NetEm as this causes packet reordering; the additional delay and jitter are thus only caused by the egress queue's size and the way the flows fill the queue. To have symmetric delays, we add half of the configured delay to each ingress of the bottleneck. While care needs to be taken in sizing the NetEm queue to not cause artificial packet loss, this approach has the advantage that the end-host stacks are not involved in the delay which is known to interfere badly with CC when Linux detects queuing pressure (TCP small queues) [Car17]. Further, in Scenario ⓑ and Scenario ⓒ, we even have no control over all end-hosts. To ensure that we can investigate RTT-fairness, we set different delays for each flow to harmonize their RTTs. To this end, we measure the *minimum* RTT through our testbed (using TCP pings) when not using any artificial delays for each flow. We then use each flow's min. RTT to configure delays such that all flows experience the same artificial min. RTT.

Limitations. Our testbed has several limitations that need consideration. We must ensure that our traffic shaper is the actual bottleneck of the path from the CP to our client. Since we do not have full control over all involved entities, we can only configure bandwidths that are sane, given our interconnection. Our testbed uses Gigabit Ethernet, and our Institute then connects via 10 Gbit/s to our University's backbone, which in turn connects to the German research network (DFN) via 40 Gbit/s which then peers at DE-CIX with all CPs investigated in this study. Thus, shaping traffic for typical end-user access links should render the bottleneck to our traffic shaper. Further, we need to artificially bump up the RTTs at least to the highest min. RTT measured. For us, the CPs typically show RTTs around 5 ms to 10 ms, which enables us to investigate a wide range of RTTs.

Additionally, to ensure repeatability and independence, we take several precautions to avoid undesired side-effects. First, to investigate the interaction of CC, the cwnd must be the genuinely limiting factor, which is why we advertise an initial flow-control rwnd of 200 segments. In the same way, we ensure that send and receive buffers are large enough to utilize the available bandwidth fully and do not introduce an

Setting	Parameter Space
Bandwidth	50 Mbit/s, 10 Mbit/s
RTT	50 ms, 100 ms
Buffer sizes	0.5×BDP, 2×BDP
Queueing discipline	drop-tail, CoDel, FQ-CoDel

Table 3.14 Study parameter space.

undesired new bottleneck. Finally, we clear all TCP caches after each measurement to ensure that cached metrics such as slow start threshold (ssthresh) do not affect future measurements (testbed only).

3.3.2.3 Parameter Space

Selecting reasonable parameters for our testbed is challenging. We must adhere to the testbed's limitations while seeking to replicate a reasonable end-user environment. Table 3.14 summarizes the parameter space which we discuss next.

Bandwidth. To ensure that the bottleneck link is within our testbed, we have to set the bottleneck link bandwidth accordingly. To identify the bandwidth provided by the individual CPs, we performed a larger number of bandwidth tests to determine which data rates are reliably offered by the different CPs. We have found the lowest data rates to be around 60 Mbit/s. Adding a safety margin, we choose 50 Mbit/s as our upper data rate limit which according to Akamai [Aka16] is representative for mid-sized access links. Further, we choose 10 Mbit/s as a lower bound to represent a low-end connection.

RTTs. We choose 50 ms as the lower bound for the min. RTT and 100 ms as a representative for higher latencies, even though usually we expect typical CPs to have much lower RTTs to their customers. However, these increased RTTs make it possible to reduce the relative error when we pad up the RTTs to ensure RTT-fairness between connections.

Buffer Sizes. For the bottleneck buffer, we experiment with different queue sizes since we know of no study that investigates typical last-mile buffer sizes. While the potential for overly large buffers (bufferbloat) is known [GN12], less than 1% of the end-user flows were observed to experience RTT variations larger than 1 s by a major CDN [HPC+14][28]. Therefore, we choose one overly large buffer in the order of 2×BDP and, inspired by research advocating new buffer sizing rules ($\sqrt{num\_flows}$ [AKM04] and $\log win\_size$ [EGG+06]), one smaller buffer size of 0.5×BDP which, for our investigated bandwidths and delays, yields queue sizes between both proposed sizing rules (see Section 2.3.4.1 for an in-depth discussion on buffer sizing).

AQM. In addition to these parameters, we also change the queuing discipline between a regular drop-tail queue and controlled delay (CoDel) [NJ12]/FQ-CoDel to investigate the impact of AQMs on fairness (see Section 2.3.4.2 for a discussion on AQMs and an explanation of CoDel and FQ).

[28] Even though we regard values far below 1 s as already bloated.

3.3.2.4 Fairness Metric

We rate the fairness by capturing the traffic that the client receives for each flow. To this end, the client requests a first flow from one machine, and after 5 s, a second flow from another machine. Both flows then continue to transmit data for another 40 s before shutting down. Of the overall 45 s, we investigate 35 s starting 5 s after the second flow starts its transmission. While this methodology is above all intended to focus on the long-term fairness between the two flows, we also examine whether it is vital which flow is started first by including experiments with a flipped starting order. We repeat each measurement 30 times to investigate the stability of our results.

To rate the fairness between both flows, we look at the ratio of transmitted bytes (over the timespan of the shorter flow) and define our fairness measure as,

$$fness(a,b) = \begin{cases} 1 - \frac{bytes(a)}{bytes(b)} & \text{if bytes(b)} \geq \text{bytes(a)} \\ -1 + \frac{bytes(b)}{bytes(a)} & \text{if bytes(a)} > \text{bytes(b)} \end{cases}$$

Intuitively, *fness*(A,B) maps the fairness behavior of the two flows into the range of [-1, 1] with zero indicating absolute fairness, -1 that Flow A absolutely dominates Flow B and a value of 1 the opposite. In between, the measure depicts the ratio of bytes actually transmitted, e.g., 0.5 indicates that Flow B transmitted twice the bytes compared to Flow A.

3.3.2.5 Testbed Validation

To investigate if our testbed produces meaningful results, we seek to confirm known findings about the behavior of CC with our testbed. Since related work considers a wide range of parameter settings and different variations of dumbbell topologies, we do not aim to exactly replicate specific results of related work, but rather general findings that are similar throughout all works. For this, we focus on Scenario (a) (testbed-only) and test whether the performance of out-of-box CC algorithms in our pure-testbed scenario is similar to the findings of related work as presented in Section 3.3.1, especially regarding inter- and intra-protocol fairness.

Figure 3.44 visualizes this measure for a configuration with 10 Mbit/s and a min. RTT of 50 ms for BBR, CUBIC, and a CUBIC when activating pacing. We show a scatterplot of all our measured values together with a kernel density estimate to better visualize the location of the majority of our measured data. For each combination of algorithms, we plot the results when Flow A starts first (yellow) side-by-side with the switched setting when Flow B starts first (violet). For the tests where a CC algorithm performs against itself, switching which flow starts first only mirrors the data at the 0-axis.

Our results show expected values as all algorithms generally show a considerable degree of fairness to themselves (intra-protocol fairness) with BBR showing a bit of a larger variance compared to the others. When comparing the *inter*-protocol fairness, we observe that BBR clearly monopolizes the bandwidth regardless of

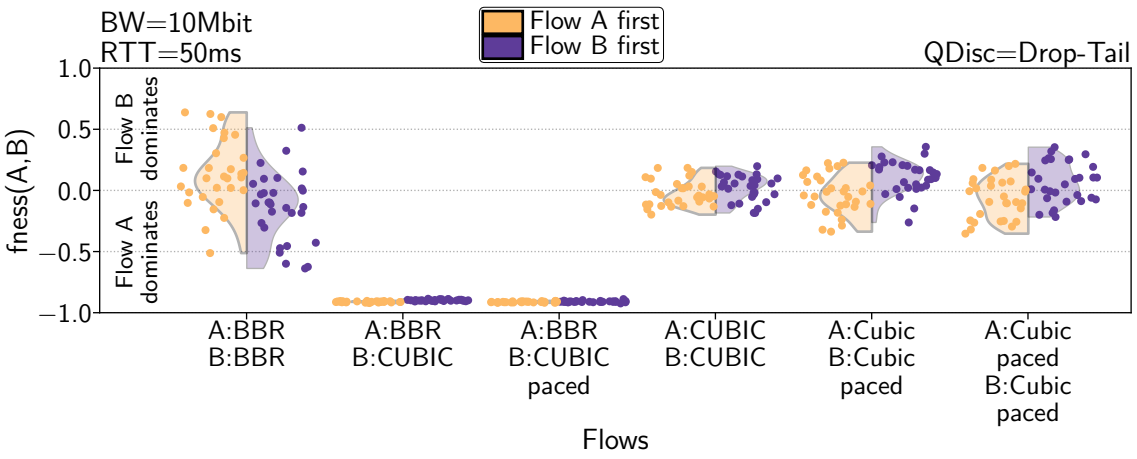


Figure 3.44 Results from Scenario ⓐ for 10 Mbit/s, 50 ms and a buffer of 0.5×BDP for different CC algorithm combinations.

which flow starts first. This observation confirms related work on BBR in low-buffer scenarios [CCG+16a, HBZ17, SJS+18]. An additional finding is that pacing seems to decrease fairness when competing with both paced and non-paced CUBIC flows.

While these experiments validate that our testbed yields meaningful results confirming known findings, we now investigate Scenario ⓑ and Scenario ⓒ to study how CPs, and thus possibly algorithms as used in the Internet, perform against our known CC algorithms and each other.

3.3.3 Congestion Control in the Wild

We base our evaluation of TCP fairness on actual Internet traffic by six major CPs (Akamai, Amazon, Cloudflare, Edgecast, Fastly, and Google) in two settings: *i)* lab vs. CP and *ii)* CP vs. CP in February 2019. Studying actual Internet traffic is motivated by the observation that CC research often neglects the complex parameterization possibilities. In Section 3.1, we found that CDNs use different IW configurations, and some utilize pacing. To this end, we suspect that not only the IWs might be different, thus choose two URLs for Akamai (named AkamaiA (which used IW32 in our previous contribution) and AkamaiE (having used IW16)) mapping to these different settings. Furthermore, Cloudflare and Google have both publicly announced to utilize BBR. Thus, we opt to observe the performance of actual Internet traffic originating from these six different CPs when competing against our testbed flows in Scenario ⓑ and among themselves in Scenario ⓒ.

We obtain URLs generating big responses (the smallest being 25 MB) served by each CP by analyzing the HTTPArchive [SGM+19]. Since the responses can still be too small to cover our 45 s measurement period, we make use of HTTP/2 multiplexing, i.e., we request the same resource multiple times (in parallel) over the same connection enabling us to prolong the transmission by a multiple of the original file size. The h2load tool shipped with nghttp2[29] already provides this functionality.

[29] `https://github.com/nghttp2/nghttp2`

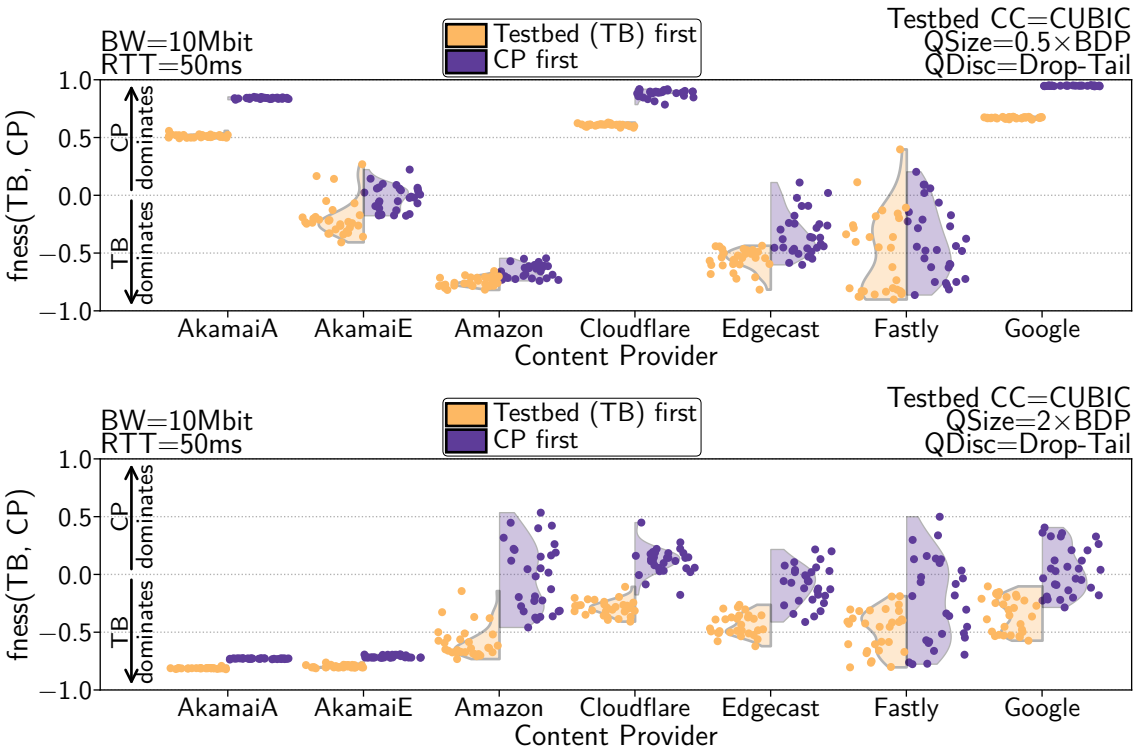


Figure 3.45 A CUBIC flow originating from our testbed competes with the CPs for traffic using a small buffer (top) and a large buffer (bottom).

3.3.3.1 Lab Traffic vs. Content Provider Traffic

We start by investigating Scenario ⓑ where traffic from our testbed machines, i.e., BBR and CUBIC flows, competes with Internet traffic and hence the algorithms employed by our CPs.

CUBIC Small Buffer. Figure 3.45 shows the results for 10 Mbit/s and a min. RTT of 50 ms when using a CUBIC flow. We again use our fairness measure to plot the measurement results; here, results < 0 indicate a dominance for our testbed flow while results > 0 favor the CP. As observed in the upper plot for measurements with a small buffer of 0.5×BDP, Cloudflare and Google definitely dominate the traffic in all instances, giving little bandwidth to our CUBIC flow (unfair setting). Apart from these two, Amazon and Edgecast struggle against our CUBIC flow even when their flow starts first (unfairness by our testbed flow). In contrast, Fastly — at least when having a headstart — can achieve rough fairness. The two Akamai flows offer a different behavior with AkamaiE showing the highest degree of fairness while AkamaiA is similar to Cloudflare and Google in that it completely dominates our testbed's CUBIC flow. This observed difference in the behavior of the two Akamai flows supports our initial guess that Akamai uses different configuration parameters.

CUBIC Large Buffer. When looking at the large buffer setting in the plot below, we observe a different picture. Now, the Cloudflare and Google flows do not dominate anymore, and the fairness heavily depends on which flow we initiated first. Similarly, for Amazon, Edgecast, and Fastly, when the testbed initiates the first flow, they

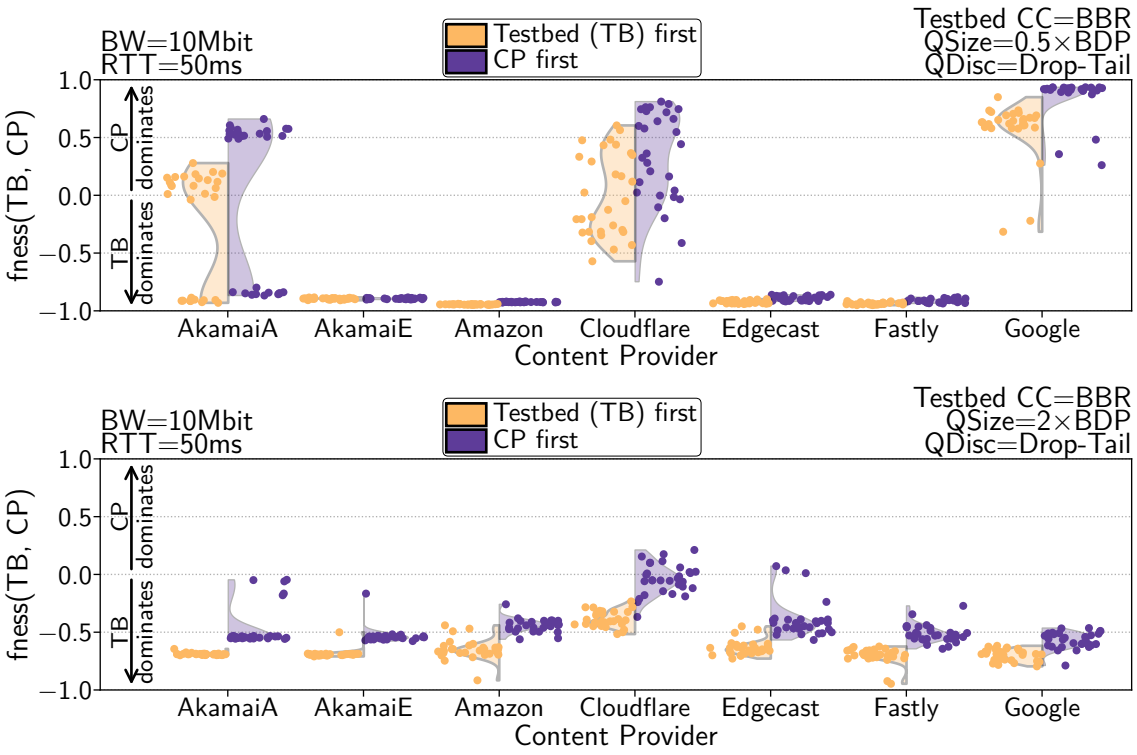


Figure 3.46 A BBR flow originating from our testbed competes with the CPs for traffic using a small buffer (top) and a large buffer (bottom).

struggle to gain enough bandwidth. In contrast, when the CP initiates the first flow, a generally fairer distribution is achieved. For Amazon and Fastly, we observe a bi-modal distribution of the traffic shares, one that more closely matches the testbed-first case and another that tends to favor the CP. What is very interesting to see is that our testbed CUBIC flows dominate both Akamai flows entirely, no matter which flow we start first.

BBR Small Buffer. Things start to significantly differ when we configure our testbed's flow to utilize BBR, as shown in Figure 3.46. As can be seen in the upper plot, showing the fairness under a small buffer setting, the testbed BBR flow dominates the flows of Amazon, Edgecast, and Fastly. The same is true for AkamaiE, while AkamaiA shows an unusual behavior. Parts of the experimental iterations show the apparent dominance of the testbed flow while about half of the iterations either show a very fair result (when the testbed flow starts first) or dominance of the Akamai flow (when the Akamai flow starts first). The observed characteristics appear to be stable in that the behavior seems to switch between two distinct states. Cloudflare shows a wide range of observed fairness ratios from dominating the testbed flow to the opposite. For Google, however, our testbed flow always clearly loses to the CP.

BBR Large Buffer. In the large buffer scenario, the flows of most CPs show very similar behavior in that the testbed flow dominates the competition. The effect is most visible when the testbed flow starts first while it is slightly ameliorated when the CP is the first flow. The degree of unfairness is thus similar across the board

	BBR@2BDP		BBR@.5BDP		CUBIC@2BDP		CUBIC@.5BDP	
	QSize	Retrans	QSize	Retrans	QSize	Retrans	Qsize	Retrans
AkamaiA	43	175	14	1244	60	22	16	80
AkamaiE	42	210	13	967	59	24	12	70
Amazon	53	468	12	836	66	30	12	31
Cloudflare	40	22	13	1319	57	40	12	166
Edgecast	53	377	12	810	64	33	12	41
Fastly	53	442	11	741	65	31	12	41
Google	40	215	15	760	55	50	14	184

Table 3.15 Average queue size (QSize) and retransmissions (Retrans) of the testbed originating flows for the 10 Mbit/s, 50 ms scenario with the testbed flow starting first.

with Cloudflare being the only real exception as it achieves a balanced fairness level when the CP flow starts first.

Retransmissions. In addition to only looking at the resulting fairness, we also consider fundamental characteristics of the bottleneck buffer and the participating end-hosts. In this case, we observe the queue size of the bottleneck buffer, which we measure using an eBPF program on the bottleneck machine, and the number of retransmissions of the testbed flow, which we also measure using an eBPF program on the corresponding testbed machine, i.e., *Testbed1* in Figure 3.43. We present these characteristics in Table 3.15. What can be seen is that for a testbed CUBIC flow, Cloudflare and Google cause significantly higher retransmission counts than the other CPs. What is very interesting is that Cloudflare induces very few retransmissions for the BBR testbed flow in the large buffer scenario but the most retransmissions in the small buffer scenario.

Higher RTT and Higher Bandwidth. When we investigate our other settings with higher RTTs, we observe no qualitative difference in fairness for all but AkamaiA. AkamaiA's bimodal fairness distribution in the 50% BDP setting shifts towards the testbed dominating all measurements. When increasing the bandwidth, the testbed BBR flow still dominates, but the fairness focusses for Cloudflare and Google, especially for smaller buffer sizes; the larger buffer generally leads to a broader distribution of the fairness. Especially, Amazon, Edgecast, and Fastly can claim slightly more bandwidth on average. Looking at changes when using CUBIC in the testbed, we observe no significant difference when competing against Cloudflare and Google. For the others, we observe a slight trend towards more bandwidth for the CPs. Again, AkamaiA stands out in the small queue setting and behaves like AkamaiE when increasing the bandwidth. We validated AkamaiA's behavior over several days (repeating the same 30 measurements for the different settings) and were able to observe the same changes consistently.

Takeaway. *As indicated by our results, fairness largely depends on the available buffer size. Generally, it seems that the CC algorithms employed by the CPs are achieving better fairness with off-the-shelf algorithms when more buffer size is available. However, large buffers can cause jitter and generally inflate the latency. In small buffer settings, BBR claims nearly all bandwidth and shows a large variability in*

fairness and performance when competing with other BBR flows causing unpredictable performance.

While it might seem advantageous at first glance that algorithms like BBR claim more bandwidth than, e.g., CUBIC, it could actually be bad for CPs. In the Web, CPs often compete with third-party resources loaded on the same website. When the CP claims all bandwidth, it may negatively affect the website loading behavior since they could cause reduced performance for the competing flows of the other resources. Thus, CPs should interact fair with their competitors, which is the focus of the next part of our study, i.e., how two CP flows interact.

3.3.3.2 Content Provider vs. Content Provider

For investigating the interaction between the different CPs, we now deploy Scenario ⓒ, where we request *both* flows from the CPs. The rest of the testbed configurations remain unchanged. Figure 3.47 shows the results for AkamaiE, Amazon, and Cloudflare flows competing against the other CPs in a scenario with 10 Mbit/s, a min. RTT of 50 ms and a small (left column) or large (right column) buffer size. Due to the similarity of the results, Amazon serves as a representative for Edgecast and Fastly, while Cloudflare represents Google as well. Once more using our fairness measure, results < 0 indicate a dominance of the explicitly mentioned CP while results > 0 favor the competing CP mentioned on the x-axis.

Small Buffers. Starting with the left column, i.e., with the small buffer scenario, we rarely observe cases where the CPs achieve a good level of fairness. Especially Cloudflare (bottom) seems to dominate most of the other CPs with the only exception being when it is forced to compete with Google and AkamaiA. In the former case, Google generally dominates Cloudflare when it starts first while we observe large range fairness results when Cloudflare is the first flow. We make the most interesting observation for AkamaiA as the bi-modal behavior observed before is again visible when it is started first and forced to compete with Cloudflare. Here, roughly half of the results indicate significant domination by AkamaiA.

For the scenarios where we focus on AkamaiE (top) and Amazon (center), it is evident that Cloudflare and Google massively dominate them. The same holds when they compete against a first flow originating from AkamaiA, while the behavior is much fairer when the AkamaiA flow starts later. When interacting, Amazon, Edgecast, and Fastly show a rather high degree of fairness. When AkamaiE competes against Edgecast and Fastly, we observe a broad range of fairness values, ranging from medium dominance of Edgecast and Fastly to total domination of AkamaiE. The latter, i.e., total domination of AkamaiE, is above all visible when competing against Amazon.

Large Buffers. Things again change when we focus on the right column using a larger buffer. Regarding Cloudflare (bottom), we observe that the strict dominance is less profound than in the small buffer scenario, yet still favoring it. When the Cloudflare flow starts first, there is a higher degree of fairness when competing against Amazon, Edgecast, and Fastly, yet they struggle when Cloudflare's flow starts first.

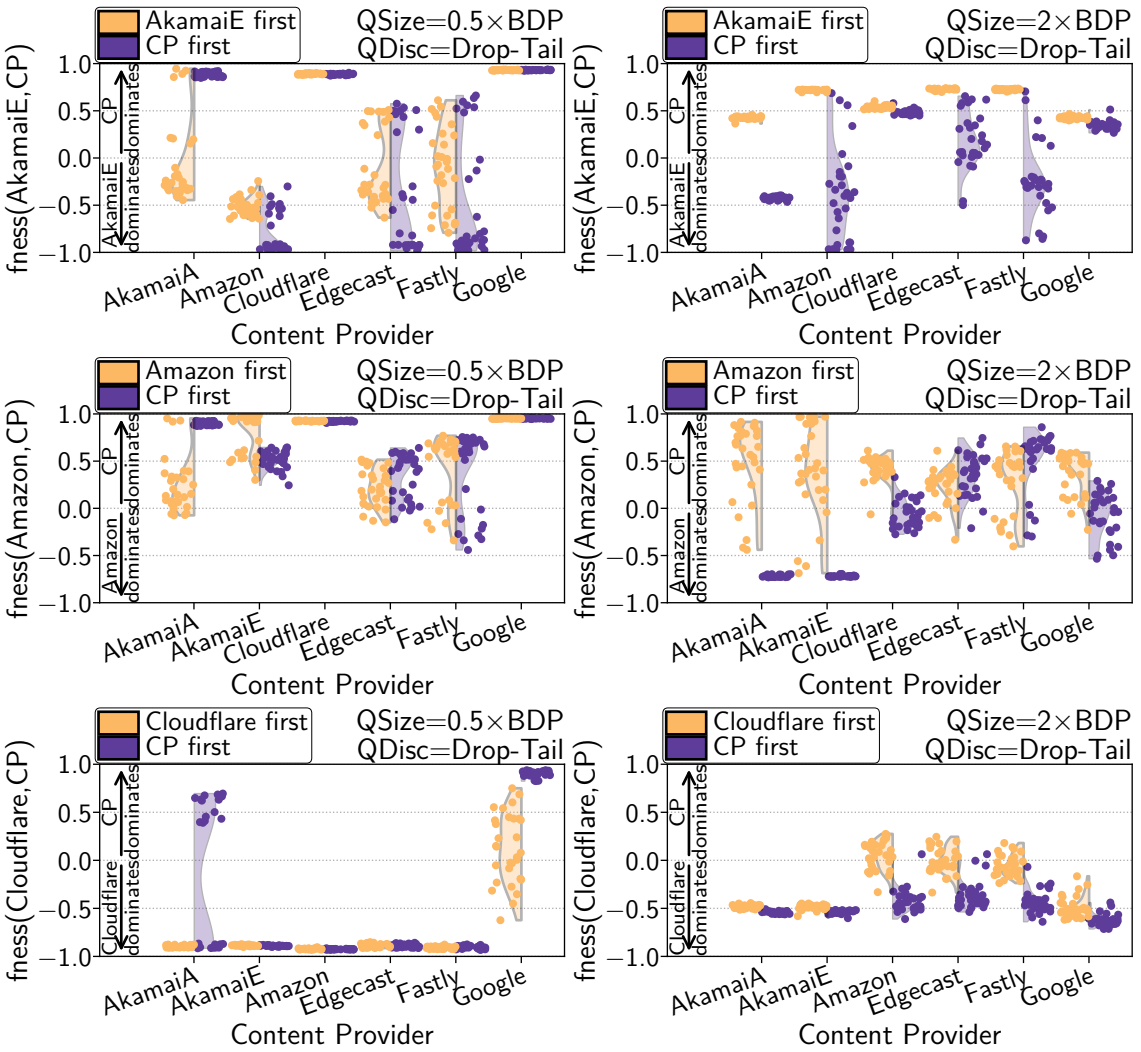


Figure 3.47 AkamaiE, Amazon, and Cloudflare competing against the other CPs in the 10 Mbit/s, 50 ms setting. Amazon performs similar to Edgecast and Fastly, Cloudflare is similar to Google. Left column shows 0.5×BDP, right column 2×BDP.

However, Cloudflare does not seem to cooperate well with the two Akamai flows or Google, as it dominates them in all these scenarios.

Generally, we witness a decent amount of fairness in several scenarios involving Amazon (center). Especially when competing against Cloudflare, Edgecast, and Google, high fairness levels are achieved. In contrast to that, we can again see very poor fairness for the Akamai flows if they start first, while we observe a wide range of values when they compete against an Amazon flow starting first. When the AkamaiE flow is the first contester (top), the other CPs dominate, yet, AkamaiE seems to be able to claim bandwidth better when it enters as the second flow.

Takeaway. *As observed earlier, larger buffers seem to enable a better level of fairness, even though they are still far from being equal in most cases. This observation especially holds for Cloudflare and Google, which dominate most of the other CPs in the small buffer scenario while there are reasonable fairness values for most CPs in the large buffer setting.*

Even though we note a higher level of fairness for the large buffer, it comes with the problem of larger queue sizes and hence also with increased delays. Ideally, we would have a scenario with a smaller queue but still the high level of fairness. As AQMs, like CoDel, are designed to keep the delay (and hence the queue) small, we are interested in whether they can help to achieve the desired combination of small delays and high level of fairness. Thus, we investigate the effect of AQMs on the whole situation in the following section.

3.3.3.3 Can CoDel Improve Fairness?

AQMs inherently change the behavior of a queue which, is why they have a significant impact on the overall performance. Generally, they have two possible forms of feedback to which flows might respond: i) dropping packets and ii) using a marking scheme such as explicit congestion notification (ECN). In our work, we only consider the first case of feedback, i.e., packet drops because it requires no end-to-end support. For this, we repeat the experiments from before but activate CoDel and its flow-queuing variant FQ-CoDel on the intermediate bottleneck machine (see Section 2.3.4.2 for a description of CoDel and FQ). In the following, we further concentrate on the case with a queue size of 2×BDP because CoDel's effect on small queues is likely to be diminishing. Hence, Figure 3.48 only shows the results for a queue size of 2×BDP in the otherwise unchanged scenarios previously used in Figure 3.47, i.e., for 10 Mbit/s and a min. RTT of 50 ms. We again choose AkamaiE, Amazon, and Cloudflare as the showcase CPs.

CoDel. The main observation that we make is that CoDel (left) seems to achieve a very high level of fairness when Amazon competes with Edgecast and Fastly and when Cloudflare competes with Google. Apart from that, there are above all awfull fairness values when other CPs are competing against Amazon or Cloudflare with tendencies looking like the small buffered scenario with the FIFO queue.

For the AkamaiE flow, the application of CoDel comes in hand with apparent domination of Akamai when competing against Amazon, Edgecast, and Fastly. What is more, Akamai also dominates Cloudflare and Google when it starts first, while there are again two regions of values when Cloudflare and Google start first; one where Akamai dominates and one where the other two dominate. Finally, when looking at the performance against AkamaiA, the bi-modal effect is again visible and now for both cases. AkamaiE starting first is again characterized by a dominance of AkamaiE half of the time and a fair behavior the other half, while this is the exact opposite if AkamaiA starts first.

FQ-CoDel. Shifting towards the flow queuing variant (right), which is designed to produce a fair queuing, we observe a tremendous increase in fairness. Now, throughout all measurements, fairness is close to the equilibrium and we only observe slight variations. When looking at AkamaiE, we see the most significant variation relative to the others with AkamaiE slightly dominating most of the others. Looking at Amazon, we see a slight advantage that diminishes for Edgecast and Fastly. In the Cloudflare case, Amazon, Edgecast, and Fastly get slightly less bandwidth while Google is very fair, and the Akamai flows again showing a slight bi-modal pattern.

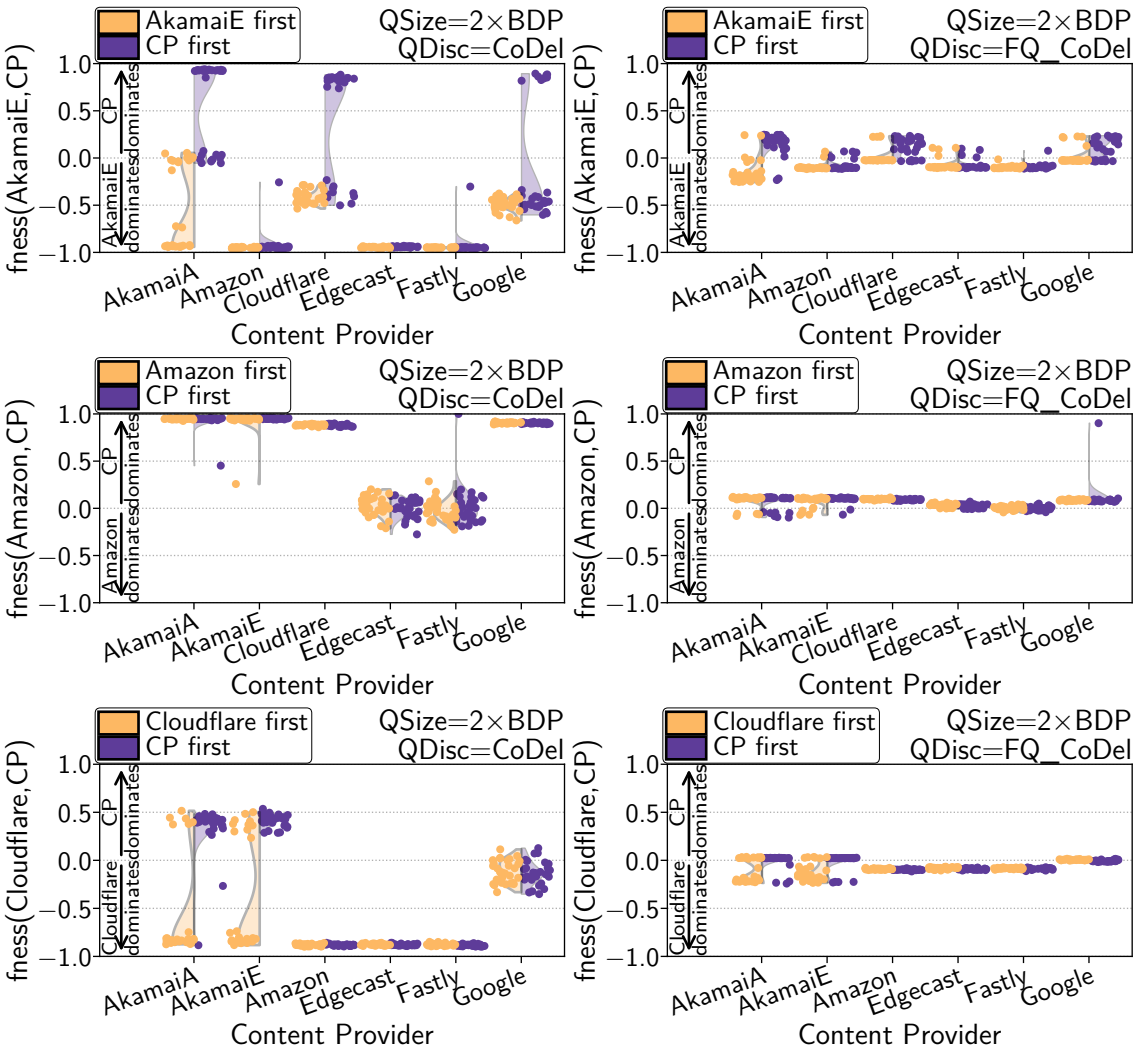


Figure 3.48 AkamaiE, Amazon, and Cloudflare competing against the other CPs in the 10 Mbit/s, 50 ms, 2× BDP setting using CoDel (left column) and FQ-CoDel (right column). Amazon performs similar to Edgecast and Fastly, Cloudlfare is similar to Google.

Takeaway. *Combining the findings on AQMs with our previous observations that Amazon, Edgecast, and Fastly use a similar algorithm and that Cloudflare and Google use BBR. We find that CoDel, above all, seems to improve the intra-protocol fairness in large buffers. This news is terrible for the heterogeneous Internet, as scenarios with different algorithms suffer from severe unfairness. Luckily, the flow queuing variant enables a significant degree of fairness even in heterogeneous settings. Thus, it seems to stand again that the technologies to enable a fair and performant Internet are available and only need to be deployed at the bottlenecks.*

3.3.4 Summary and Discussion

In this contribution, we empirically investigated the fairness of large content providers in the Internet. With the help of our testbed, we can investigate actual Internet traffic subject to RTT-fairness when competing under lab-controlled properties of a

bottleneck. Generally, we find there is only limited fairness in the Internet today. Some CPs interact well with each other, while others do not. Their choice in CC algorithm likely reflects this observation. We thus posit that increased research in the area of CC identification is required, that not only detect the flavor of CC, e.g., CUBIC or Reno but further investigate their parameterization. Further, we find that the bottleneck buffer size significantly impacts the fairness, depending on its size, it can invert bandwidth sharing observations when shifting from small towards large. This finding demands research to investigate actual bottleneck buffer sizes in the Internet to then shine a light on, e.g., the impact on Web performance when content is served from a diverse set of CPs. Still, there is a silver lining: State-of-the-art AQMs such as FQ-CoDel put the fairness control back into the network operator's and possibly the end-user's hand, however, require availability and deployment on millions of devices.

3.4 Conclusion

In this chapter, we have focussed on the impact of Internet giants on Internet transport. To this end, we have illuminated how the Internet has evolved and is currently operated through three lenses.

First, we showed how TCP, as the de-facto transport of the Internet since the 1980s, is being used today at large by investigating how IWs are configured in IPv4 and by Internet giants. Our measurements have shown that RFC-recommended values are on the rise at large but that Internet giants such as content delivery networks that carry the bulk of the Internet customize this highly influential parameter. We found instances that exceeded the current experimental recommendation by a factor of ten, which may sound dangerous at first sight. However, our measurements and evaluations also showed that CDNs are well aware of the effects that their changes have, many putting safety measures such as pacing in place. Further, our measurements even showed indications for per-network and per-service customization, showing that these values outside of the recommended ranges are an explicitly defined choice.

Our measurements highlight how and that evolution today is still possible in the Internet at large with a protocol that is considered highly ossified. Within this first contribution, we overcame several measurement challenges, such as determining the quantities in which we must scan the Internet to derive results regularly while reducing the footprint of our scans. We further demonstrated the ability to leverage the distributed nature of VPNs to measure from various network locations enabling us to investigate how residency within a particular network affects our measurements and the results.

Second, we investigated the deployment and use of QUIC, as an alternative and highly anticipated transport protocol that combines many improvements to TCP while enabling an ossification-free deployment. While promising increased efficiency through deployable transport optimizations, network operators struggle with QUIC as QUIC hides the once passively accessible transport headers through encryption which

challenges their TCP-dependent network management. With our measurements, we provide the first broad analysis of QUIC in the wild by enumerating QUIC hosts in all of IPv4 and roughly 50% of the DNS namespace and by analyzing various traffic traces. We find that the originally proposed Google-version of QUIC was initially mainly deployed by Google, but Akamai as a major CDN now operates the most extensive publicly reachable gQUIC infrastructure. Similar to our first contribution, we were able to see a significant degree in parameterization of different gQUIC deployments which even allowed fingerprinting them. By further analyzing the current IETF draft versions of QUIC, we found that many more players are now involved, yet, again Internet giants such as Cloudflare or Facebook dominate the deployment.

In contrast to this, our analysis of QUIC traffic shares at a major European IXP and ISP showed that deployment and traffic shares are highly asymmetric and vantage point dependent. We find that Google transports over half of its traffic already via QUIC and in total, we observe QUIC traffic peaks of up to 20.0% at the ISP. Surprisingly, we find little Akamai-originating QUIC traffic given their extensive infrastructure, even though they push more traffic via QUIC at the IXP than Google. Our findings hint at Akamai still testing QUIC on a small subset of their customers. Given the large efforts that are put into QUIC, we were surprised that our performance evaluations showed that QUIC is not the clear winner when it comes to convincing actual humans. Our studies showed that QUIC excels beyond TCP when the networks are prone to loss, but when the networks offer high bandwidth and low latency, QUIC can only win by a small margin on paper but humans do not perceive a difference for website delivery. Still, even though we believe that deploying QUIC might thus not be everyone's top priority, some of QUIC's features are simply not deployable in TCP at large and QUIC's design itself offers the possibility to rapidly evolve which shows a clear path beyond a TCP-based Internet.

Our insights highlight that QUIC already challenges operators today and the infrastructures that are in place for gQUIC and that are currently set into place for iQUIC have the potential to shift significant amounts of Internet traffic from TCP to QUIC. Further, we have witnessed a vibrant QUIC landscape with quickly evolving versions. While most of these versions are used to test or introduce certain features, QUIC, by-design, allows for this rapid evolution. We thus ask whether, in the future, ossification will happen on the end-points that become stale and that do not install software updates offering new versions or if the future Internet will become a rugged landscape offering various QUIC versions. We found that Internet giants such as Google that control the whole ecosystem, i.e., browser, service, and serving infrastructure, can quickly push changes without consensus and are thus likely not affected by end-host ossification. Less popular services that do either not possess the same infrastructure or will not update may seem faced with radical development cycles by Internet giants. However, our analysis also showed that today, it takes an Internet giant such as Google or Akamai, to push such fundamental transport changes as QUIC to make them applicable at large and to become an Internet standard that has the potential to become an Internet reality.

Third, while pushing innovations to the IETF for standardization, our analysis on IWs has already shown how Internet giants operate outside of these standards in a pursuit

for increased performance and hence user satisfaction. Therefore and motivated by the increased future ease to change CC in QUIC, we investigated how the traffic of these Internet giants still adheres to the general notion of fairness through the eyes of a regular home user. As the Internet, by design, is a *decentralized and uncontrolled* system no measures are currently put into place to monitor or enforce fairness of the participating systems, even though it is known that (US) mobile operators do rate-limit individual CPs [LNC+19]. We designed our measurement study to find out if CC, as implemented by the Internet giants, conforms to lab-controlled behavior of CC algorithms generally considered fair. To this end, we ensure RTT-fairness by design and observe great variations of flow-rate fairness across the board. While we find that some Internet giants generally show approximate fairness to one another, others largely dominate testbed as well as traffic originating from their competitors.

Generally, this sounds bad at first glance. However, Internet flows have varying durations (e.g., bulk download vs. website delivery), phases of inactivity (e.g., video streaming) and often depend on one another (e.g., website resources that are subsequently discovered and fetched). Thus, claiming that one Internet giant falsely takes resources from others away is too short-sighted, it may even be non-beneficial to steal resources in case of a website delivery when critical third-party resources are thus slowed down. Even though our measurements uncovered pathological unfairness, the nature of the traffic *may* render this unfairness non-existent and further, research has devised methods such as FQ-CoDel that can be applied decentralized by everyone to reestablish a fair bandwidth allocation that guarantees low latencies for all involved parties.

In summary, Internet giants today coin practical Internet transport. The sheer dominance in terms of bytes and services that they deliver puts them into a powerful position. Them flipping a switch can change how significant parts of the Internet and especially the Web are operated. While they generally open up their innovations and are keen to standardize them, the configurations, especially for congestion control, are a business secret and, e.g., IWs are vastly different among the large players. Thus, we find that Internet reality has drifted away from standardized and taught practice and that research needs to investigate the nature of real Internet traffic regularly.

We are going to continue our view on Internet evolution by focussing on the ossification and deprecation on the network layer and how individuals can regain control of content routing that is currently dominated by Internet giants.

4

Evolution in the Internet's Core

In the previous chapter, we have observed that the ossification on the transport layer challenges Internet evolution and that configurations alone are no more sufficient for large Internet giants, and new protocols are set in place to overcome these hurdles. While middleboxes challenge these end-host only changes, evolution in the Internet's core is even more challenging by design as all systems involved need to be updated. Even though in the early days of the Internet such a change was still easy given the small number of participating systems,

> *" [...] after a short grace period of a few months, no network was allowed to participate in the Internet if it did not comply with IPv4."*
>
> — *Leonard Kleinrock [Kle10]*

Today, this evolutionary challenge is largely reflected in the ongoing Internet Protocol (IP) transition from IP Version 4 (IPv4) to IP Version 6 (IPv6) that is going on since 1998. Figure 4.1 shows the share of Google users contacting Google services via IPv6 between September 2008 and August 2019. While this is of course only Google's view and does likely over-emphasize eyeball networks, it still shows that the transition is happening but that it took roughly until 2013 to gain some traction.

While our focus is not on IPv6 in this chapter, it shows how long such a fundamental change takes, and that evolution in the Internet's core is a slow process. In this chapter, we thus tackle the question of *how do content owners flexibilize in light of Internet giants and network ossification?* Today, Internet giants heavily peer with other systems to be less dependent on public infrastructure and to serve content with low latencies. This resulting high quality of service has led to a large dependence on them and is forcing individuals to apply their practices.

Our approach to this is thus two-fold, Section 4.1 first shines a light on network ossification and deprecation. We do so by listening to Internet control plane feedback

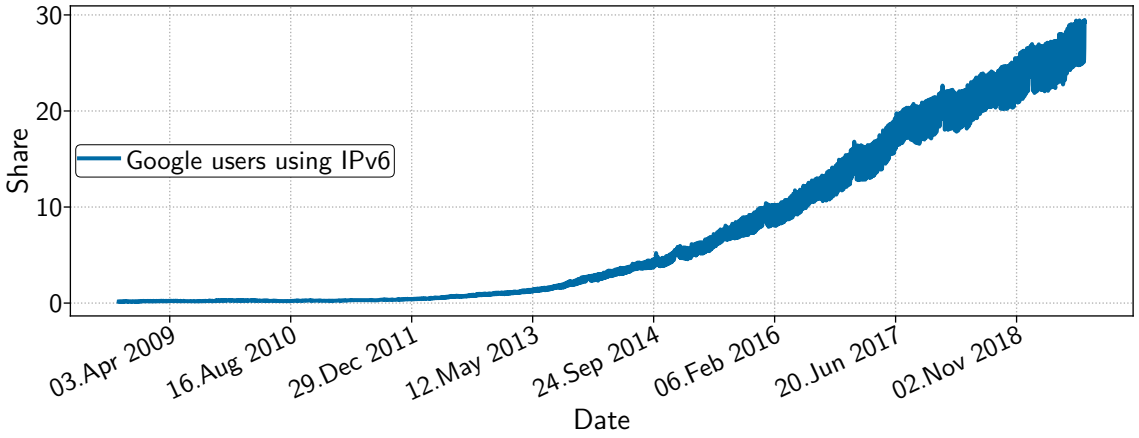


Figure 4.1 IPv6 adoption among users using Google services, data available at [Goo19a]. Please note the visually thick appearing line is due to a heavy fluctuation caused by a weekly pattern where Google sees more IPv6 usage on Fridays, Saturdays and Sundays.

in response to the large-scale Internet measurements from Chapter 3, thereby recycling the measurements and lowering the footprint of Internet scanning by not having to perform additional scans. Then, Section 4.2 analyzes how content owners practically circumvent network ossification and the restrictions and regimentations set in place by Internet giants. We do so by dissecting one of the largest Meta-CDN and shining a light on their operational model, thereby documenting how content provision can be liberated. For our measurements, we need to overcome several research challenges.

Research Challenges

- **How to study Internet ossification on a large scale?**

  The Internet is an extensive collection of systems that interconnect and interact with each other. Measuring its evolution at large is challenging, especially given the large number of possible ways to test evolution and the large number of IP addresses. Thus, our methodology should have a low footprint to be ethical. Furthermore, this then raises the question of how (un)-evolved the core is and whether network operators regularly update their systems to follow standardization and current best practices.

- **Meta-CDNs are highly distributed, and their customer base is unknown.**

  Measuring highly distributed systems such as content delivery networks (CDNs) is challenging from a single vantage point. It requires a global perspective from many networks to derive the operational model of a CDN. Thus, our methodology needs to be executable on a wide variety of platforms. Further, companies are usually restrictive in giving out information regarding their users, but we nevertheless want to establish who uses Meta-CDNs and for what kind of service.

4.1 Listening into the Void – Studying Internet Core Evolution

As we have seen in the previous chapter, Internet scans are a valuable and thus widely used approach to understand and track the evolution of the Internet. To this end, we have already applied them to a significant extent measuring large parts of the Domain Name System (DNS) as well as the public IPv4 address space. We are not the only ones conducting network scans, they are applied in widely different fields, including networking and security research: e.g., to find vulnerable systems [DLK+14], to measure the liveness of IP addresses [BRJ+18], or to, as we also did, measure the deployability of new protocols, features [EKT+17], or their evolution [VSN+16]. The recent advancements in scanning methodologies enabled probing the entire IPv4 address space for a single port within minutes or hours, depending on the available bandwidth and configured scan rate (see tools such as ZMap [DWH13] or MASSCAN [Gra19]). Thereby, regular scans of the entire IPv4 address space have become feasible, e.g., providing an insightful perspective into protocol evolution (see, e.g., our QUIC measurements in Chapter 3). This line of scan-based works has created a rich body of contributions with valuable insights into Internet structure and evolution. These works have in common that they focus on one particular feature or protocol as their objective to study (*primary use*).

To now study Internet core evolution, we argue that Internet-wide scans have a less explored *secondary use* that allows studying the Internet control plane while scanning for their primary use, e.g., to detect routing loops while *primarily* probing for QUIC-capable servers. That is, we study Internet control plane responses sent via the Internet Control Messaging Protocol (ICMP) as a response to non-ICMP probe packets (e.g., QUIC) and show that Internet-wide scans are a hidden treasure in that they produce a rich ICMP dataset that is currently neglected, e.g., to uncover network problems and study Internet evolution. The exiting aspect is that these ICMP-responses are a valuable secondary use that any Internet-wide scan generates as a by-product. Thus, without putting additional burden on the Internet's infrastructure, we can study the Internet control plane (e.g., to detect routing loops) without requiring dedicated scans (as performed a decade ago [HMM+02, XGF07]).

Our observations on the Internet's control plane are fueled by regular ZMap scans of the IPv4 address space for multiple Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) ports as well as DNS-based scans of top lists and zone files for mainly Transport Layer Security (TLS), Hypertext Transfer Protocol (HTTP)/2, and QUIC. Most of which we have presented in Chapter 3. In this contribution, we evaluate one full week of ICMP responses to these multi-protocol Internet-scans covering the entire IPv4 address space and $> 50\%$ of the domain name space.

Our contributions are as follows:

- We propose to use Internet-wide scans to study the Internet control plane via ICMP response, e.g., to detect routing loops or to study the prevalence of misconfigurations and misuses that have been documented long ago thus shedding light on Internet core evolution.

Mon	Tue	Wed	Thu	Fri	Sat	Sun
			DNS-based TCP/443 gQUIC/443			
Alexa 1M TCP/80 TCP/443	1% IPv4 TCP/80 TCP/443	IPv4 TCP/80	IPv4 gQUIC/443	IPv4 iQUIC/443	IPv4 TCP/443	

Table 4.1 Weekly scan schedule fueling our dataset, DNS-based scans use our own resolver infrastructure. For IPv4-wide scans, we utilize ZMap.

- Within our one week observation period, we collect ∼637.50M ICMP messages which we make available at [Rüt18d].
- We shed light on how Internet-scans trigger ICMP responses across the Internet.
- Our data shows a plethora of misconfigured systems, e.g., sending ICMP redirects across the Internet or producing deprecated source quench messages.
- We find many networks and hosts to be unreachable, and our scans uncover large sets of unreachable address space due to routing loops.
- We provide a growing ICMP dataset at `https://icmp.netray.io`.

Structure. The next section (Section 4.1.1) starts by providing an overview of our ICMP dataset. Following this, we dive into our dataset and dissect it (Section 4.1.2). Driven by our findings, we inspect unreachable hosts due to routing loops and quantify their presence in today's Internet (Section 4.1.3). Finally, we discuss related works (Section 4.1.4) and conclude this contribution (Section 4.1.5).

4.1.1 Scan Infrastructure & Dataset

Our scans are sourced by two different modes, on the one hand, we use the ZMap [DWH13] port scanner on multiple machines to perform different scans within a week, and on the other hand, we continuously probe $> 50\%$ of the DNS space. Table 4.1 summarized our weekly scan schedule. We did not explicitly create these scans to serve this effort; in contrast, they are used to fuel the data that we have presented in Chapter 3 and other ongoing research efforts.

These scans typically involve scanning TCP/80 for TCP initial congestion window (IW) configurations (from Section 3.1) or TCP Fast Open support (see Chapter 3). Further, we investigate TCP/443 for HTTP/2-support [ZRW+17] and TLS [HHA+20]. Additionally, we scan on UDP 443 for Google QUIC (gQUIC) and Internet Engineering Task Force QUIC (iQUIC) (see Section 3.2). We drive our DNS-based scans with the help of our own resolver infrastructure to resolve various record types for domains listed in zone files of multiple top-level domains (TLDs) (e.g., .com, .net, .org), which we obtain from the different registries, and we use A-records to investigate TLS, HTTP/2, and gQUIC. All our scans, including the DNS resolutions, originate from a dedicated subnet that otherwise does not generate any eyeball traffic. To collect all ICMP traffic that is directed towards these hosts, we install a mirror port at

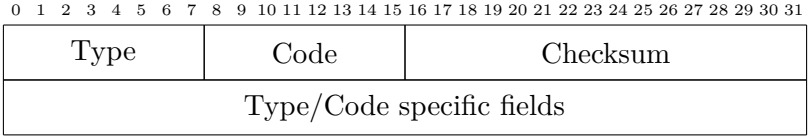


Figure 4.2 ICMP header structure. Type and this type's sub type (code) determine message contents, e.g., often packets triggering the ICMP message are quoted.

Type	Count	Unique IPs	Unique ASes
Destination Unreachable	476.68M	170.30M	52.92K
Time Exceeded	139.53M	455.13K	18.40K
Redirect	18.12M	243.25K	2.29K
Echo Request	3.12M	10.64K	861
Source Quench	46.18K	2.65K	364
Echo Reply	6.08K	301	58
Other	1.48K	606	43
Timestamp Request	73	9	6
Parameter Problem	20	16	9
Address Mask Request	4	1	1

Table 4.2 ICMP types with their occurrence frequency in our dataset, ordered by frequency.

their uplink switch and filter it to only contain ICMP traffic that belongs to our measurement network. Since we perform no measurements that generate ICMP messages themselves, we exclude those sent from our host (only ping responses) leaving us with only incoming ICMP traffic.

Dataset

We base our observations on one full week in September 2018. In this week, we received 169 GB respectively ∼637.50M ICMPv4[30] messages (excluding those explicitly triggered in Section 4.1.3). ICMP messages follow the structure shown in Figure 4.2, they are fundamentally made up of a type field and, to further specify a subtype, a code field, and depending on their value additional information may follow.

4.1.2 Study of ICMP Responses

To begin our investigations, we first summarize the ICMP responses to our scans by looking at the distribution of ICMP message types and their frequency of occurrence in Table 4.2. We observe 75 different ICMP type/code combinations during our observation period with significantly different occurrence frequencies. While we mostly receive standardized ICMP messages, we also receive some messages for which we could not find an Request for Comments (RFC) or other documentation, summarized as *Other* in Table 4.2, on which we do not further focus in this contribution. The

[30]Please note that we do not have a fully IPv6-capable measurement infrastructure and thus focus on IPv4 only.

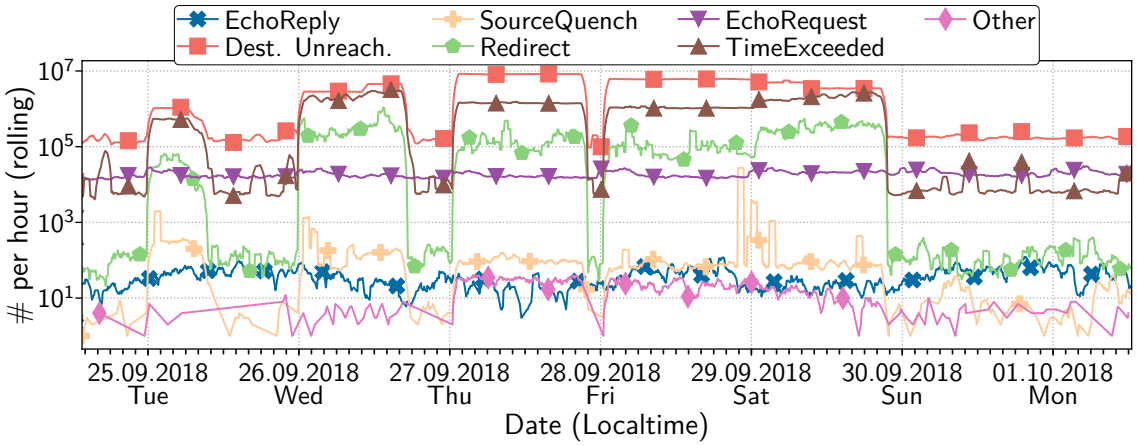


Figure 4.3 Number of ICMP messages receiver per hour and type over the course of a week. Note the log scale and that we used a rolling sum over 1h.

table lists the total count of these messages as well as the number of unique source IP addresses (router/end-host IP addresses) that generated the messages and number of autonomous systems (ASes) where the systems reside. During the week, we run different scans, notably, on Sundays and Mondays (see Table 4.1), we do not perform any IPv4-wide ZMap scans.

Figure 4.3 thus puts the data from Table 4.2 into a temporal context showing the rolling sum over 1h intervals of the most occurring ICMP types. We observe that the ICMP traffic varies over the week, e.g., echo requests are somewhat static, other types like destination unreachable mainly follow our ZMap scan schedule.

Quoted IP Packet

Apart from the different ICMP types, many ICMP messages contain parts of the packet that caused the creation of the messages. We further inspect these quoted IPv4 packets within the ICMP messages. From all received ICMP messages, 99.5% are supposed to contain IP packets (according to the RFCs), of these, only 0.07% cannot be decoded, e.g., because there is not enough data or they are not IPv4 packets. Of the decodable packets, we find 180.25M unique source IP address/payload length combinations, 76% are longer than 40 B, i.e., enough to inspect IP and TCP headers when the packets contain no options[31], 24% are precisely 28 B long, so just enough to inspect the transport ports. Thus, when no options are present, the chances are high that an ICMP receiver can demultiplex the ICMP messages to the respective application process. This finding extends the work in [ML07] that showed a prevalence of 28 B responses for TCP `traceroutes`. Next, we focus on the destination address field within the quoted IP header. These should correspond to addresses which are targeted by our scanners.

Interestingly, from all ICMP messages, we find over 1.06M messages with destination IP addresses that are in reserved address space, i.e., unallocated or private addresses

[31] To reduce the capture size, our packet capture caps packets at 98 B allowing no further investigation, we find 67% having this maximum capture size.

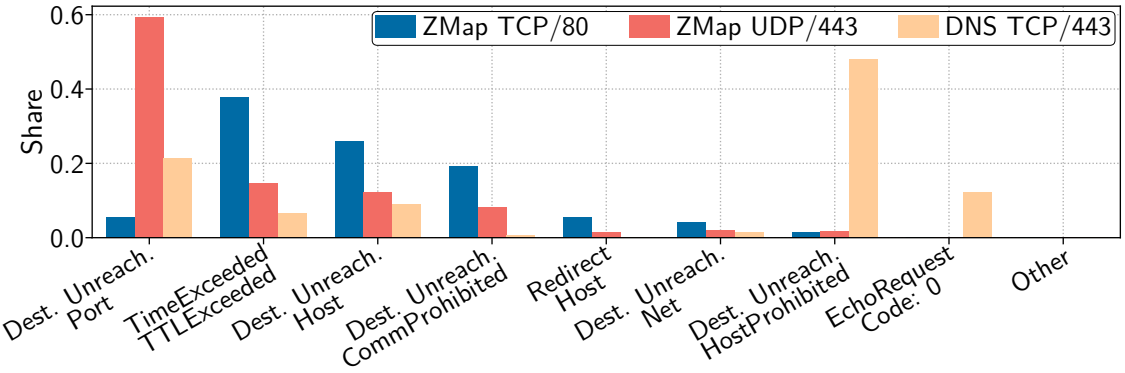


Figure 4.4 ICMP messages triggered by ZMap and DNS-based scans.

(e.g., 192.168.0.0/24). Since all our scanners explicitly blacklist these IP addresses, we thus believe that machines behind network address translations (NATs) produce these messages. We next use the contained source addresses to understand the relation to our measurements.

Takeaway. *ICMP traffic shows a temporal correlation to measurement traffic; most messages indicate unreachability. In our collected dataset, quoted IP packets typically contain enough information to inspect everything up to the end of the TCP header. Further, a substantial number of messages seems to be generated behind NATs allowing to peek into private address spaces.*

4.1.2.1 Responses to Individual Measurements

Since we perform a variety of different measurements independent of this study, our first investigation is how different measurements affect the generation of ICMP traffic. To this end, we compare two ZMap scans and a purely DNS-based scan. For the ZMap scans, we focus on one that enumerates reachable TCP port 80 (HTTP) and UDP port 443 (QUIC) hosts, for DNS, we use a scan that probes for HTTP/2 support via TCP port 443. We can unquestionably tie the ICMP messages to the different scans via IP address and ports either from the quoted IP message or from the IP packet itself.

Figure 4.4 shows the distribution of ICMP types *and* codes (top 8) that we receive for the respective scans. As already indicated by Table 4.2, we receive a large number of destination unreachable messages. However, depending on the scan, their volume and share greatly vary, especially when looking at the respective code. For example, unreachable ports are omnipresent for our UDP-based ZMap scan. In comparison, the TCP-based ZMap scan shows only a small fraction of unreachable ports. This finding is unsurprising as TCP should reply with an RST-packet if a port is unreachable and does typically not generate ICMP messages. In contrast, there is no such mechanism in UDP, even through something comparable to TCP's RST exists in QUIC. However, QUIC is implemented in user space, and thus, when the kernel cannot demultiplex a packet to a socket, it must resort to issuing an ICMP unreachable message. Looking at our DNS-based scan, we still find that more than

20% of the ICMP messages signal unreachability through ICMP in contrast to TCP RSTs, something that, e.g., the default ZMap TCP-SYN scan module ignores in contrast to its UDP counterpart. Since in all major operating systems, TCP handles signaling closed ports, we believe that these hosts issuing ICMP replies are actively configured either in their own firewalls (e.g., iptables) or in a dedicated firewall to do so. We find only 16.49K IPs issuing *all* 1.13M ICMP port unreachable messages, supporting our assumption that dedicated machines filter this traffic.

Looking at the other types/codes, we find that a non-negligible share of ICMP messages indicate that hosts are not reachable via the Internet either due to time to lives (TTLs) expiring or because we cannot reach their host or network. Apart from this, we observe that TCP port 443 is often firewalled (Host Prohibited).

Takeaway. *Depending on the protocol and port, we get different feedback from the Internet's control plane. Our findings indicate that, e.g., ICMP port unreachable messages should not be ignored for TCP-based scans as is currently the case.*

4.1.2.2 ICMP Echos

ICMP echo requests (Type: 8) are the typical `ping` to which hosts answer using an echo reply. [RFC792] defines only a single code point, i.e., code = 0 which represents "no code"; still, we observe some non-standard code points. Some security scanners use non-standard code points for operating system fingerprinting, e.g., a standard Linux will echo the requested code point in its reply. Still, pings to our measurement infrastructure seem quite common, for code = 0, we find 10.57K unique IPs out of 840 ASes. It seems that our scanning activities trigger systems to perform ping measurements towards us, yet, we do not know their actual purpose. We suspect that intrusion detection systems (IDSes) could monitor the liveness of our hosts and thus cause the pings.

Echo Replies

Since our hosts do not perform echo requests, we were surprised to find echo replies in our dataset. We observe different code points with different frequencies, but overall, we find over a couple of thousand of these replies. To investigate what causes these seemingly orphaned messages, we inspect their destinations. Since our measurements are identifiable either by IP address or additionally by weekday, we associate messages to measurements. We find most echo replies are with code = 3 (except for five messages), all 5.75K of these echos are destined to our DNS resolvers and originate from only 86 IP addresses in two Chinese ASes. While many ICMP packets contain IP quotations, echo replies typically do not. They usually mirror data contained in the echo request, yet, we still find IP packets together with DNS query *responses* that are destined to our resolver. Thus, it seems that the packets are generated on the reverse path. However, they are not sent back to the source (DNS server), but they are forwarded to the destination (us). Inspecting the source IP within the IP fragments, we find IP addresses from the same two ASes, as it turns out the 88 ICMP source IP addresses all respond to DNS queries which hints at their

use as a DNS server cluster. Nonetheless, we were unable to manually trigger these ICMP reply packets when trying to send DNS requests to these IP addresses. We only observed that DNS requests were always answered by two separate packets from the same IP address, however, with different DNS answers. Further, the packets seem to stem from different IP stacks (significantly different TTLs, use of IP ID or not, don't fragment bit set or not). While the different stack fingerprints could be the result of middleboxes altering the IP headers, the general pattern that we observe hints at DNS spoofing.

4.1.2.3 Source Quench

ICMP source quench (SQ) messages (Type: 4, Code: 0) were a precursor of today's explicit congestion notification (ECN) mechanism, used to signal congestion at end-hosts and routers. The original idea from [RFC792] was that a router should signal congestion by sending SQ messages to the sources that cause the congestion. In turn, these hosts should react, e.g., by reducing their packet rate. However, research [Fin89] found that SQ is ineffective in, e.g., establishing fairness and the Internet Engineering Task Force (IETF) deprecated SQ-generation in 1995 [RFC1812] and SQ-processing in 2012 in general [RFC6633]. Major operating systems ignore SQ-messages for TCP at least since 2005 to counter blind throughput-reduction attacks [RFC5927]. Further, [Flo94] claims that SQ is rarely used because it consumes bandwidth in times of congestion.

In our traces, we observe 2.65K unique IP addresses located in 364 ASes issuing SQ messages, despite the deprecation. Out of these IP addresses, 34.42 % are located in only five ASes. Moreover, our measurement infrastructure contacted 609 SQ-generating IP addresses directly, i.e., they are the original destination of the request causing this SQ message (according to the IPv4 header contained within the ICMP message). Among the remaining SQ messages, we find a few messages where the original destination and the source of the SQ messages are located in ASes of different operators, i.e., possible transit networks. Exemplarily, we observe that IP addresses located in AS1668 (AOL Transit Data Network) and AS7018 (AT&T) issued SQ messages when we contacted IP addresses located in AS8452 (Telecom Egypt). As a final step, we see that 53 destination IP addresses in our measurements trigger the generation of SQ messages and are also contained in A-records of our DNS data that we collect. Out of these 53 IP addresses, 22 IP addresses generated the SQ messages themselves, i.e., no on-path intermediary caused the creation of this message.

Besides, we checked how vendors implement or handle this feature. Cisco removed the SQ feature from their IOS system after Version 12 in the early 2000s [Cis08]. Hewlett Packard's cluster management system (Serviceguard) generated SQ messages due to a software bug in a read queue, which was fixed by a patch in 2010 [Hew10]. In their router configuration manual (September 2017), Nokia also marks SQ messages as deprecated [Nok17]. Although we cannot identify devices and their operating system version in our measurements, we assume that some devices are not updated to a current version or are following a configuration that enables them to generate SQ messages. This generation is not forbidden per se, but given that ICMP SQ creation

was deprecated over 20 years ago, our findings highlight that removing features from the Internet is a long term endeavor.

4.1.2.4 Redirect

ICMP redirect messages (Type: 5), are sent by gateways/routers to signal routes to hosts. While [Gil02] finds networks which require redirect messages to be architected sub-optimally in the first place, [RFC1812] states that a router *must not* generate redirect messages unless three properties are fulfilled: *i)* The packet is being forwarded out the same physical interface that it was received from, *ii)*, the IP source address in the packet is on the same logical IP (sub)network as the next-hop IP address, and *iii)*, the packet does not contain an IP source route option. Similar checks [RFC1122] are used by receiving hosts to check the validity of the message (e.g., redirected gateway and issuing router must be on the same network).

Since none of the 18.12M redirect messages originate from our network, the routers generating them either violate rule *ii)* or some obscure address translation is in place on their networks. In our data, we even find roughly 2.7K unique redirects to private address space. Within our dataset, we observed 105.78K network redirects and 18.01M host redirects. Network redirects are problematic since no netmask is specified and it is up to the receiving router to interpret this correctly. For this reason, [RFC1812] demands that routers *must not* send this type. We find that the network redirects originate from 238 different ASes affecting nearly 19K different destinations of which less than 20 are mapped in any of our DNS data. Still, all these ASes thus contain questionable router configurations that are outdated at least since 1995. Similarly, we find that the more considerable fraction of host redirects originate from 2.20K ASes that affect over 400K destinations of which we find roughly 900 mapped in our DNS data. This conjuncture suggests that a substantial number of end-systems connect via sub-optimally architected or misconfigured networks to the Internet.

4.1.2.5 Unreachable Hosts

Reachability is a fundamental requirement to establish any means of communication. Given that Table 4.2 lists 476.68M destination unreachable messages, this looks troublesome at first. However, not all unreachability means faulty behavior, e.g., firewalls actively protect infrastructure from unpermitted access, i.e., when iptables *rejects* a packet (in contrast to dropping it) it generates an ICMP response. By default, a port unreachable message (Type: 3, Code: 3) is produced, but a network operator can manually specific other types. Our scans in themselves certainly trigger a particular number of firewalls or some IDSes. In contrast, when a path is too long and the IP TTL reaches zero, routers typically generate an ICMP TTL exceeded message indicating that they were unable to reach the destination but this time due to the network's structure. Similarly, ICMP destination unreachable messages for host unreachable (Type: 3, Code: 1) should indicate that there is currently simply

Type	Code	Count
Destination Unreachable	Port	256.72M
Time Exceeded	TTL Exceeded	139.52M
Destination Unreachable	Host	107.15M
	Comm Prohibited	71.70M
	Host Prohibited	23.07M
	Net	17.94M
	Protocol	51.04K
	Fragmentation Needed	26.66K
	Net Prohibited	26.28K
Time Exceeded	Fragament Reassembly	7.31K
Destination Unreachable	Host Unknown	336
	Net TOS	25
	Net Unknown	6
	Source Isolated	2

Table 4.3 ICMP messages received indicating some form of unreachability with known type and code ordered by frequency.

no path to a host, e.g., because it is not connected or the link is down. Table 4.3 summarizes the unreachability that we observe in our dataset.

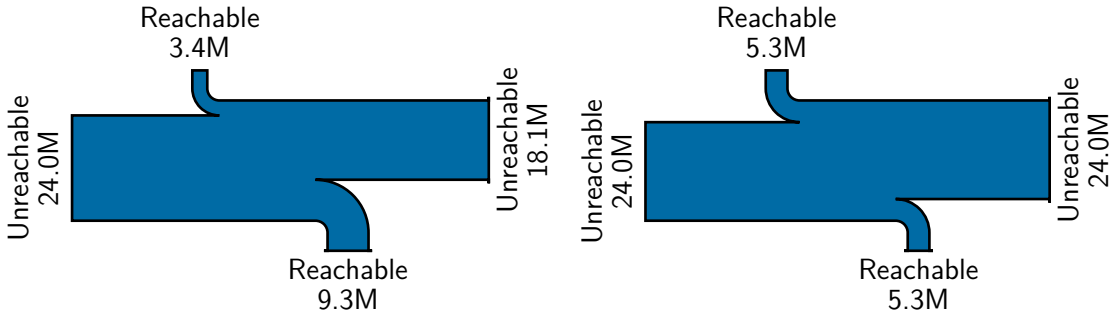
As already indicated in Section 4.1.2.1, our UDP-based ZMap scans have the highest share of port unreachable messages putting them at the top. We inspect the origin of the messages and the actual destination that our scans targeted to see if the end-hosts generate the messages or an intermediate firewall. It seems that 96% of the messages are indeed generated by end-hosts or machines that can answer on their behalf (NATs).

Host and Network

Unreachable hosts and networks codes are used to give hints that currently no path is available and the RFCs explicitly note that this may be due to a transient state and that such a message is no proof of unreachability. To check for transient states, we compare the unreachable hosts on Thursday with those on Friday in our ZMap (both UDP 443) scan and additionally with the same scan (Thursday) one week later (captured separately from our initial dataset) and investigate if hosts become reachable that were unreachable before or vice versa.

Figure 4.5 visualizes the change between these two days (a) and within one week (b) for host unreachable messages. We can see that within two days, the majority of hosts remain unreachable, a small number of hosts that were previously *reachable*[32] become unreachable, and similarly, previously unreachable hosts become reachable. Looking at the changes within a full week, we observe that the total number of unreachable hosts stays the same. However, roughly the same number of previously reachable

[32]With reachable we actually mean *not unreachable*, i.e., we do not get ICMP unreachable messages, which must not mean that this host was reached by the scan.



(a) Thursday to Friday. (b) Thursday to Thursday one week later.

Figure 4.5 Different scans (left to right of each plot) trigger different number of host unreachable messages. (a) compares the changes within one day. (b) within one week.

hosts become unreachable and vice versa. To dig into these once unreachable and then reachable hosts, we inspect to which AS they belong, finding that 82% of all hosts are from the same ASes. A possible explanation might be that while our observations seem to indicate a change, the ICMP message generation is subject to rate-limiting [GH18]. Thus there might be routers that generated unreachable messages on Thursday for a certain host, yet, this router could be subject to rate-limiting on Friday for the same host or the week after leading to a false impression of reachability and continuity. Another possibility is that some hosts are only up at certain times of the day leading to differences in the reachability. Still, a substantial number of hosts remain unreachable.

Time Exceeded

Similar to host unreachability, Time Exceeded messages (Type:11) indicate unreachability but due to network issues. Either the Fragment Reassembly (Code: 1) time was exceeded, i.e., the time that IP datagrams are buffered until they can be reassembled when IP fragmentation happens, or the TTL runs out (Code: 0), i.e., the path length exceeds the sender-defined limit. For the former, we find some thousand messages, but they stem from only 30 ASes, since many of our scans use small packets, fragmentation is unlikely in the first place. However, our UDP ZMap scans are an exception, since they roughly use 1300 B per packet which is in the range of typical [CFL18] maximum transmission units (MTUs) when fragmentation could occur. Since the default ZMap functions to create IP packets (which we use), do not set the *don't fragment bit*, only some of our measurements trigger the 26.66K *fragmentation needed and DF set* ICMP messages (see Table 4.3). However, over time, these ICMP messages could give valuable insights into path MTU in the Internet.

TTL Exceeded messages have the second-largest occurrence (139.52M) within our dataset. They were produced in 18.40K different ASes covering 35.5M different destinations that our scans tried to reach of which ∼32K are again present in A-

records of our DNS data and are thus unreachable. We inspect the TTL field of the quoted IP packets that triggered the ICMP messages to see if the TTL was really zero when the message was generated. To do so, we first generate all unique pairs of router IP and TTL values and then count the different TTLs observed. Out of these, 97% of the TTLs show a value of one, followed by ∼2.4% with a zero, we expect these two, since a router should drop a TTL = 0 or, depending on the internal pipeline, also TTL = 1, when the packet is to be forwarded. Nevertheless, we also find larger TTLs, 2, 3, 4, 5, and 6 directly follow in frequency, yet, we also find some instances of over 200 or even 255. The huge TTLs could hint at middleboxes or routers rewriting the TTL when they generate the message to hide their actual hop count. The lower numbers could be indicators for Multiprotocol Label Switching (MPLS) networks. By default, e.g., Cisco [Cis14] and Juniper [Jun17] routers copy the IP TTL to the MPLS TTL on ingress and also decrement the IP TTL within the MPLS network. It is possible to separate IP TTL and MPLS TTL, and there are heated discussions whether one should hide the MPLS network from traceroutes or not, which has also been the subject of investigations [DLM+12]. Thus packets expiring within an MPLS network will still trigger an ICMP TTL exceeded, however, the quoted IP packet will have the TTL value they had at the MPLS ingress router, thus if the ingress copies the IP TTL, a traceroute could still reason about an MPLS network.

Since we were surprised to see this many TTL exceeded messages across all scanner types (see Section 4.1.2.1), we checked our scanners to see which TTL they were using to see if our setup simply has too small values. All our ZMap-based scanners initialize the TTL field with its maximum of 255 possible hops, all scanners building on top of the transport layer interfaces, in contrast, use the current Linux default of 64 hops as also recommended in [RFC1700]. Given that we are at least on the recommended hop count, this leaves us with three possibilities, *i)* the current recommendation of 64 is too low to reach these hosts, *ii)* there are middleboxes modifying the TTL to a much lower value, or, *iii)* there are routing loops on the path to these hosts. After shortly summarizing our findings, we continue by exploring the latter.

4.1.2.6 Summary

As the previous sections have shown, our Internet-wide scans produce an insightful *secondary* dataset of ICMP responses. Driven by these messages, we identified a potential DNS spoofer, found that long deprecated source quench messages are still generated in today's Internet and that ICMP redirects are sent across different administrative domains pointing to several outdated and misconfigured networks. Without crafting a dedicated dataset, our scans enable us to study Internet reachability and evolution, and we believe that longitudinal studies offer a way to deal with the challenge of ICMP rate-limiting.

4.1.3 Routing Loops

Routing loops are an undesirable control plane misconfiguration, rendering destination networks unreachable and challenging a link's load [XGF05]. In essence, IP's TTL protects the Internet from indefinitely looping packets and thus ICMP TTL messages inform the sender that a router dropped a packet after exceeding the allowed number of router hops (TTL). While the potential for routing loops is known, only a few studies investigated their presence a decade ago [HMM+02, XGF07], current information on the presence and prevalence is missing. Therefore, we study routing loops based on ICMP TTL exceeded messages triggered by our scans. We further argue that one can frequently investigate the presence of routing loops as a by-product of Internet-wide scans that one regularly conduct for different purposes.

4.1.3.1 Methodology: Detecting Loops

Loops are not the only source of ICMP TTL exceeded messages. Also, overly long paths or middleboxes can trigger these messages. To investigate whether or not an actual loop is present, we perform `traceroutes` for the original destinations (given in the quoted IP packet) of the ICMP TTL exceeded messages. Since our traceroutes are subject to ICMP rate-limiting, especially when packets start to loop, we customize traceroute. Our traceroute slows down its sending rate when detecting an already seen IP address (loop indicator). Otherwise, it follows the design of *Paris traceroute* [ACO+06] reusing flow identifiers for each hop to trigger the same forwarding behavior in equal-cost multi-path (ECMP)-like load balancers.

Since the traceroutes can still be noisy due to hosts that do not generate ICMP messages at all or are still subject to rate-limiting, notably when also other traffic flows into a loop, we put strict demands on our loop. For each hop on the path that does not generate a reply, we assign a new unique label; we label all others directly by the answering IP address. From this list of labels, we create a directed graph connecting each label-induced node to its successor and, on this path, we compute all elementary cycles using [Joh75]. On an elementary cycle, no node appears twice except that the first and last node are the same. Then, on each of these possible cycles, we inspect the node with the highest degree, and if this node's degree is greater than five[33], we mark this traceroute as having a loop. This process will yield loops as long as at least one router in the loop generated ICMP TTL exceeded messages, which we found to work reasonably well when traceroute pauses the packet generation for at least 500 ms when observing an already seen IP address. Thus in a loop of two routers, we will send each router a packet roughly every second.

4.1.3.2 Routing Loops in the Wild

We seed our traceroutes by ICMP TTL exceeded messages generated from our Internet-wide scans[34]. Since we get way too many TTL exceeded messages to

[33] This is basically a precaution against bad load balancers traded against the required TTL.
[34] Our dataset excludes TTL exceeded messages generated by these traceroutes.

traceroute them all without generating substantial rate-limiting, we restrict us to a single traceroute for each unique /24 subnet within 30-minute intervals. Thus for two TTL exceeded messages for a destination from the same /24 subnet, we only perform a single traceroute if the messages arrive within 30 minutes.

For our assessment of routing loops, we investigate TTL exceeded messages in the last week of August 2018. To avoid rate-limiting, we also limit our traceroutes that we perform in parallel; generating all traceroutes for this single week took us until the end of September 2018. While this skews our data, it enables us to reason about the persistence of these loops since, in principle, we could schedule a rescan of the same /24 every 30 minutes (subject to an ICMP TTL exceeded message appearing from one of our regular scans). In total, we performed ∼27M traceroutes to ∼612K different /24 subnets from 28K ASes, of these, 439K subnets from 19.8K ASes are unreachable due to a loop. We further inspect how many loops are present and if loops are only within a single AS or whether loops cross AS borders and are thus potentially on a peering link. To do so, we count the number of distinct loops and ASes involved in the loops and find 167K different loops in 13.9K ASes. Of these loops, 136K have IPs for all routers involved in the loop, thus allowing an in-depth inspection. Looking at the ASes involved, we find that 13% (17.7K) already cover all different ASes paths involved (i.e., we replaced each IP address by the respective AS), of these 4.8K cross AS boundaries. The top three ASes involved in the loops are AS171 (Cogent) a Tier-1, AS9498 (BHARTI Airtel Ltd.), an Indian Internet service provider (ISP), and AS3549 (Level 3), again a Tier-1.

Persistence

To investigate the persistence, we restrict our view to traceroutes, which were performed two weeks after our Internet-wide scans that triggered the initial ICMP TTL exceeded message. In contrast to our previous observation, loops from roughly 150 ASes disappear, yet, we continue to find 4.6K loops crossing AS borders, in total, still rendering 404K subnets unreachable. Thus, most loops seem to persist and are not resolved.

Loops at our Upstream ISP

Within our data, we also found loops in the AS of our upstream ISP. We contacted the ISP about our findings which they were able to confirm. Since many of the loops are outside of their administrative domain even though they manage the address space, they were still able to give us more details on a loop that they were able to fix. For one loop, they found that the first router had a static route for our tested destination towards its next hop, yet, the next-hop had no specific forwarding information for this destination and thus used its default gateway, which however was the previous router with the static route thus causing the loop.

Takeaway. *Routing loops seem to persist in large parts of the Internet, challenging the question if the address space cut off by the loops is in use after all or if other routes would be taken from different vantage points. We believe, when exploited,*

routing loops have considerable potential for causing congestion and thus persistent monitoring seeded by large-scale Internet measurements that informs operators could be a long-term attempt to reduce routing loops.

4.1.4 Related Work

Our work relates to approaches analyzing ICMP traffic and its generation in general, as well as approaches that focus on particular studies built upon ICMP, e.g., path/ topology discovery and routing loops. In the following, we discuss similarities and differences to our work, but we remark that the body of works building on top of ICMP is far larger but conceptually differ in that they do not analyze ICMP as a by-product.

Bano et al. [BRJ+18] also use ZMap and capture *all* (cross-layer) responses to probe traffic to infer IP address liveness but run specific measurements to generate this traffic. We believe that our dataset could be used to perform a similar analysis. Malone and Luckie [ML07] analyze the correctness of ICMP quotations. They base their analysis on a dataset obtained via `tcptraceroute` in 2005, targeting around 84K Web servers. While most of the reported messages are of type ICMP time exceeded, they also find around 100 source quench messages, which were already deprecated then. As we have shown, by looking at the ICMP responses to Internet-wide scans, we can update their findings regularly without having to craft a dedicated dataset. Guo and Heidemann [GH18] present FADER, an approach to detect the presence of ICMP rate-limiting in measurement traces. While we did not focus on rate-limiting, we found indicators for rate-limiting. We believe that longitudinal studies seeded by Internet-wide scans can, in the long run, help to overcome limited visibility due to rate-limiting.

In 2002, Hengartner et al. [HMM+02] have characterized and analyzed the presence of routing loops in a Tier-1 ISP backbone trace. Xia et al. [XGF05, XGF07] have further tracerouted over 9M IP addresses to find routing loops in 2005. Transient routing loops have also been subject to investigation [WQG+09], and they are well studied [SMD03, FB07]. Lone et al. [LLK+17] investigate routing loops in CAIDA data to study source address validation but do not focus on their prevalence in the Internet, further, in contrast to using the CAIDA dataset that actively runs traceroutes against all /24, we utilize indications from ongoing measurement data to investigate loops. While these works show that routing loops are a known problematic misconfiguration, their presence in the Internet has not been analyzed for over ten years. By recycling Internet-wide scans, we can seed such investigations and enable persistent monitoring of this phenomenon, showing that routing loops are still a problem today.

4.1.5 Summary and Discussion

In this contribution, we focussed on Internet core evolution. To reason about its current state, we recycled measurements from Chapter 3 and studied ICMP messages

generated in response to these scans. Our analyses of different ICMP responses led us to many misconfigured routers, e.g., sending ICMP redirects across the Internet, or outdated systems, e.g., generating long-deprecated source quench messages. While we find that there are a large number of systems that are outdated or misconfigured, only a small fraction of these systems seems to be on the path to DNS-mapped destinations. For the Web and many publicly used services that seems to hint that there are no major misconfigurations on the path, yet in the Internet's core at large, we still find these systems.

Further, our analysis showed a broad and nuanced degree of unreachability in the Internet. More specifically, our scans hint at the existence of routing loops, which we found to persist in large parts of the Internet and even at our upstream ISP. Routing loops are a significant misconfiguration of the control plane and can challenge the availability of a link when malicious actors send packets towards such a loop. We hope that these ICMP by-products are analyzed by more researchers when performing large-scale measurements and that the regular nature of these scans will enable persistent monitoring of the Internet's control plane and that, especially when brought to the attention of operators, misconfigurations can be fixed. To this end, we make our dataset publicly available at [Rüt18d].

Our findings highlight that the Internet core evolves slowly and misconfigurations and outdated behavior are still to be expected. This slow evolution makes it especially difficult for individuals participating in the Internet to rely on the existence of new technology, or to be able to change operation at all. In light of this logjam, we next investigate how content owners can regain control over content routing while utilizing state-of-the-art technologies such as CDNs.

4.2 Individualism in the Age of Giants – Indirection through Meta-CDNs

CDNs have become a critical key component of the Web [AGH+12, CFK+15]. Their ongoing quest to serve Web content from nearby servers has evolved the Internet structure through a hierarchical flatting [LIM+10] that promises lower latencies, while their distributed nature pledges high availability. These benefits led to a wide adoption of CDNs for Web content delivery manifesting in high traffic shares: for example, more than half of the traffic of a North American [GD11] or a European [PFA+10] ISP can be attributed to few CDNs only. In that regard, our measurements from Section 3.2 have shown that Akamai delivers a substantial amount of traffic at our European ISP. Despite these benefits, customers of a single CDN are bound to its cost model and performance figures. To this end, e.g., Netflix initially delivered its video data using commercial CDNs, but given their growth, the costs of using CDNs exceeded the costs of operating their own CDN which they announced in 2012 [Net12]. Today, Netflix is an Internet giant, but also back then had an enormous impact, yet, not every content owner is as powerful as Netflix and can design, implement, and operate its own CDN. Limitations that individual content owners can overcome by multihoming content on different CDNs and subsequently serving it from the CDN that currently offers better performance or lowest costs.

To better utilize content-multihoming, *Meta-CDNs* [FPL+13] enable content providers (CPs) to realize custom and dynamic routing policies to direct traffic to the different CDNs hosting their content; a concept also known as CDN-Selector [XCW17] and that is related to auction-based CDN brokers [MBM+16, MBR+17]. The Meta-CDN performs *request routing* according to *custom routing logic* defined by CPs (i.e., the customers of a Meta-CDN and CDNs). A broad range of factors can inform this routing logic, including CDN cost models or measured CDN performance. CPs can thus utilize a Meta-CDN to reduce costs or to optimize performance, e.g., by implementing custom logic to direct traffic to a CDN that currently offers better performance or lower cost (e.g., at certain geographic regions or times). Hence, a Meta-CDN frees CPs of the strict operational models of a single CDN and helps them to overcome limitations set in place by networks in certain geographical regions. Since the routing approach employed by the Meta-CDN customers is unknown to the involved CDNs, directed traffic and, thus, generated revenue get harder to predict. In particular, since decisions can be based on active performance measurements by the Meta-CDN, a (single) delivery of lousy performance by the probed CDN can result in rerouting traffic to a competing CDN and thus losing revenue. Thus, while Meta-CDNs can offer cost and performance benefits to CPs, they also challenge CDN business models. Concerning Internet-users, performance-based routing decisions can yield better Internet performance and benefit end-users, while cost-based decisions can have other effects (as for any server selection approach run by CDNs). While the concept is known and related work covering service-specific implementations, e.g., Conviva's streaming platform [Con19, DSA+11, MBM+16], exists, the empirical understanding of a generic Meta-CDN and its operation in practice is still limited. We posit that this understanding is necessary.

In this contribution, we thus shed light on the Meta-CDN operation by dissecting the Cedexis Meta-CDN as a prominent example that is used by major Internet companies such as Microsoft (Windows Update and parts of the Xbox Live Network), Air France, and LinkedIn [Cit18]. Given its current adoption, understanding its functionality and its usage by customers provides a first step towards understanding currently unknown implications of Meta-CDNs on Internet operation. We thus investigate the infrastructure and services powering this Meta-CDN and provide insights about its operation in practice. We analyze for *what* kind of services, e.g., media, application programming interface (API) backends, or bulk data transfers, customers utilize Cedexis and *how* different CDNs are employed. We further investigate how the infrastructure deployed by Cedexis impacts the overall request latency performance in a PlanetLab and Ripe Atlas measurement. Specifically, our contributions are as follows:

- We characterize Cedexis, as a representative generic Meta-CDN, present its operation principles, and further analyze and classify its customer base. Moreover, we illustrate which CDNs are used by the customers.
- We utilize globally distributed vantage points, i.e., Ripe Atlas, PlanetLab, open DNS resolvers, and a small deployment of probes behind home user Digital Subscriber Line (DSL) connections, to obtain a *global* view on Cedexis. Based on these measurements, we analyze the deployed infrastructure of Cedexis and are further able to investigate if the selection process varies based on the location. Besides, we find cases of suboptimal routing in terms of latency.

Structure. The remainder of this section is structured as follow, Section 4.2.1 regards related works before we dig deeper into the architecture and technical implementation of the Cedexis Meta-CDN in Section 4.2.2. Section 4.2.3 shines a light on how Cedexis' customers utilize the service from a global perspective, and finally, Section 4.2.4 concludes this contribution.

4.2.1 Background and Related Work

To achieve high availability, content, and service providers typically employ CDN operators and utilize their already deployed and geographically distributed infrastructures [AGH+12, CFK+15]. In addition to increased availability, end-users profit from the distributed nature of these CDNs when retrieving content from close-by servers, reducing the overall latency. See Section 2.1.2 for a detailed description of CDNs.

Many works from academia and industry have investigated these infrastructures, the operation principles, as well as the performance of deployed CDNs [AGH+12, CFK+15, NSS10, OSR+12]. Besides understanding and measuring CDN infrastructures, researchers have utilized CDN routing techniques to derive network conditions [SCK+09]. In addition, approaches that optimize the routing of user requests to the respective servers within a CDN, as well as, optimized anycast load balancing

have been proposed [CST15, FMM+15]. Poese et al. [PFA+10] present and analyze the impact of an ISP recommendation service, providing insights about the current network state, e.g., topology, load, or delay, to the CDN, which in turn bases its server selection on the returned information. Frank et al. [FPL+13] revisits the ideas and concepts of the presented approach, and among other features, enables a CDN to allocate server resources within the ISP on-demand when necessary.

To further ensure the availability of content, customers may use multiple CDN deployments. Other reasons to utilize more than one CDN provider may be cost efficiency, e.g., different prices to serve content at different times or due to traffic volume contracts. However, with multiple locations at different CDNs serving the same service or content, either the customer or an additional service has to choose between the actual CDN when a user requests a service or content [FPL+13, LWY+12, XCW17]. Concerning *multi-homed* content, i.e., content that is distributed by multiple CDNs, Liu et al. [LWY+12] present one of the first frameworks optimizing performance and cost of the resulting CDN assignment. In the case of video streaming, Conviva [Con19] uses a recommendation system that the video player software utilizes [DSA+11] for the CDN selection. Besides Conviva, commercial solutions that offer to act as the CDN selector in more general settings, e.g., websites or services, exist [Cit18, Ora19]. However, there is currently little to no understanding of their infrastructures, customers, and the effects on the global CDN landscape. With respect to Meta-CDNs and especially Cedexis, Xue et al. [XCW17] are the first to provide brief performance figures about the selected CDNs, focusing on deployments in China. We aim at more broadly characterizing Cedexis as a whole while looking at their infrastructure and performance on a global scale.

Nevertheless, we find, similar to Xue et al., partly suboptimal performance in terms of latency, yet, we acknowledge that routing decisions may have other goals than latency. Mukerjee et al. reinforce this argument in [MBM+16], which is closest to our work. They analyze the effect of brokers, i.e., CDN selectors, on CDNs, characterize potential problems and propose a new interface between these brokers and CDNs. While a closer interaction may improve certain aspects, it remains open whether a Meta-CDN such as Cedexis does harm a CDN's profitability. Our results do not suggest that a broker *might* prefer specific CDNs in certain regions, as we find similar CDN choices worldwide.

The goal of this contribution is to extend the currently limited understanding of Meta-CDN operation by characterizing Cedexis as a prominent example of a generic Meta-CDN. Exemplified by understanding its overall deployment, customers, and the effects Cedexis, we aim to provide a stepping stone towards a better understanding of Meta-CDNs in general.

4.2.2 Characterizing a Meta-CDN

The general motivation behind a Meta-CDN is to enable custom and dynamic routing of requests to content that is multi-homed in different content distribution infrastructures (CDIs). A CDI can involve any infrastructure ranging from simple

```
$ dig www.accorhotels.com

...

;; QUESTION SECTION:
;www.accorhotels.com.       IN  A

;; ANSWER SECTION:
        ①                                       ②    ③
www.accorhotels.com.  3210  IN  CNAME  2-01-2770-000c.cdx.cedexis.net.
2-01-2770-000c.cdx.cedexis.net.  16  IN  CNAME  cs893.wac.edgecastcdn.net.
cs893.wac.edgecastcdn.net.  2809  IN  A  152.195.39.57
```

Figure 4.6 Exemplary output of `dig` resolving a customer domain managed by Cedexis. The requesting user is redirected by the Cedexis authoritative DNS with a CNAME to the CDN selected to handle the request. The Cedexis CNAME contains the customer ID ② (C_{ID}) and a per-customer application ID ③ (App_{ID}).

(cloud-hosted) servers to complex CDNs [PFA+10]. Multihoming content on different CDIs enables CPs to optimize for availability, performance, or operational costs. By utilizing a Meta-CDN, CPs can realize *custom* routing logic to direct traffic to the available CDIs. Such custom routing logic can be motivated by CDIs that offer better performance or lower costs in some geographic regions, at certain times of the day, or when exceeding predefined request volumes. We refer to an infrastructure that enables routing between CDIs with customer-provided routing logic as a *Meta-CDN*, a concept that is also referred to as Multi-CDN selector [XCW17] and has similarities to auction-based CDN brokers [MBM+16]. Since the individual routing approaches employed by CPs at the Meta-CDN are unknown to the involved CDIs, directed traffic, and thus generated revenue gets harder to predict. In particular, since decisions can be based on active performance measurements by the Meta-CDN, a (single) delivery of suboptimal performance by the probed CDI can result in rerouting traffic to a competing CDI and thus losing revenue. While the effects of Meta-CDN operation are relevant to Internet operation, little is known about Meta-CDNs.

To elucidate Meta-CDN operation, we start by characterizing Cedexis as a prominent example. We base this characterization on showing *i)* its operational principles to select and routing between CDIs based on the Cedexis site (Section 4.2.2.1) and *ii)* its current use in the Internet by analyzing its customer base in Section 4.2.2.2 based on our measurements. Both perspectives provide a first understanding of the principle mechanisms with which Meta-CDNs can influence the content distribution and their current deployment in the wild.

4.2.2.1 Operation Principles

Like many other CDNs, Cedexis employs a DNS-based redirection scheme, similar to Akamai [NSS10], to redirect the requesting user to the CDI selected for content delivery. This redirection scheme bases on canonical name (CNAME)-records which

transfer the requesting user between the different authoritative name server (NS) (see also Section 2.1.2). We exemplify this scheme in Figure 4.6. Starting at the original domain ①, the user gets transferred to the Cedexis NS, which then selects a final CDI. The configured static Cedexis CNAME includes a Cedexis customer ID (C_{ID}) ② and configuration specific (App_{ID}) ③. Both identifiers enable the Cedexis NS to perform customer-specific request routing, once the client's DNS resolver contacts the NS for name resolution. Similar to classic CDNs, routing can be subject to a user's location, e.g., identified by DNS resolver IP address or EDNS0 client subnet extension. The Cedexis NS then points to the selected CDI, which can be an IP address in an A resource record or another CNAME, e.g., pointing to CDN (Verizon's Edgecast in this particular example). The selected CDI can then repeat this process to select the final server handling the request or point to another CDI. By realizing routing in the DNS, Cedexis redirects requesting users to a CDI *before* they establish a connection to the CDI. This way, it is not involved in the actual content delivery itself and thus does not alter the performance or security properties provided by the selected CDI.

CDI Selection Options

The above-stated request routing approach can be arbitrarily dynamic, i.e., the user to CDI mapping in the DNS can change at any time, only limited by the cacheability of their DNS records. This aspect is utilized to enable CPs to realize custom routing logic within Cedexis using three components: *i) Openmix* enables Cedexis customers to configure individual routing logic. Customers can choose between `optimal round-trip time (RTT)` to select the CDI with the lowest RTT to the requesting user, `round-robin` balancing between all CDIs configured by a customer, `throughput` to select the CDI with the highest throughput, a `static routing` or by executing `customer-provided code`. This routing logic can be informed by *ii) Radar*, an extensive database for decision making based on active browser-based CDN measurements performed by website visitors, and *iii) Fusion*, to retrieve data from CDNs. Every customer can configure site-specific behavior (i.e., *Apps*), which results in different App_{IDs} in the DNS. This way, a customer can configure different routing profiles for *downloads.domain.tld* and for *images.domain.tld*. We next describe the two data sources and the Openmix platform to realize custom routing decisions.

Radar

Cedexis provides a community-driven CDN performance measurement platform called Radar. Radar employs active measurements performed within the Web browser of visitors of Cedexis-managed websites. The in-browser measurements require the Cedexis customers to embed a JavaScript in their website. Once visited, the Web browser triggers the website's `onLoad` event, the embedded JavaScript waits for a user-configurable timeout (default 2 s) and starts requesting probe instructions from Cedexis. Users can configure private probes (i.e., to estimate their own performance) and can choose to activate community probes (i.e., enabling Cedexis to measure other infrastructures). Probes can serve different purposes, latency or throughput

measurements, which Cedexis realizes by instructing the JavaScript to fetch and measure the download time of a small 43 B file (for latency) or 100 kB file (for throughput) via HTTP. After having performed the measurements, the JavaScript reports the obtained values back to Cedexis such that they can later be used to guide performance decisions in the CDN selection process and to inform site operators about their site's performance.

Fusion

While Radar enables to realize performance-based routing decisions, Fusion enables to accomplish decisions on statistics directly from a user's CDNs. CDNs typically offer statistics about traffic shares, quotas, budgets, performance and other key performance indicators (KPIs) in their Web interfaces. By accessing these, Cedexis enables their customers to draw not only performance decisions but also business decisions (e.g., *will I hit my quota soon?*).

Openmix

Both Radar and Fusion data are available to customers in the Openmix platform. Openmix enables customers to customize the DNS resolution process by providing custom JavaScript code that the Cedexis NS executes in the DNS resolution step. Within this code, customers can define their subsequent candidate CDI choices and request measurement data (e.g., availability, latency, or throughput) for these. While many probes power the system, it returns only single values, suggesting that Cedexis preprocesses the measurement data, yet we were unable to find more information on this process. Thus, Openmix is used to realize customer-specific routing decision performed within the DNS resolution, i.e., directing traffic to a target CDI via a DNS CNAME.

Takeaway. *Cedexis offers its customers to realize site-specific, fine-granular, and dynamic traffic routing to CDIs, e.g., based on customer-provided code and informed by rich measurement data. The performed traffic routing is hard to predict (e.g., for CDIs).*

4.2.2.2 Customers

Before we analyze the infrastructure and configuration of Cedexis, we want to shed light on their customer base (in anonymous form). We are interested in which companies and businesses leverage this additional service on top of traditional CDIs.

DNS Measurement Methodology

Our approach is twofold. First, we leverage the encoded customer and application IDs in the CNAME structure (see Step ② & Step ③ in Figure 4.6) to *enumerate customers*. Applications are used by customers to define different routing profiles that map to the available CDIs, so, e.g., a customer may have one profile

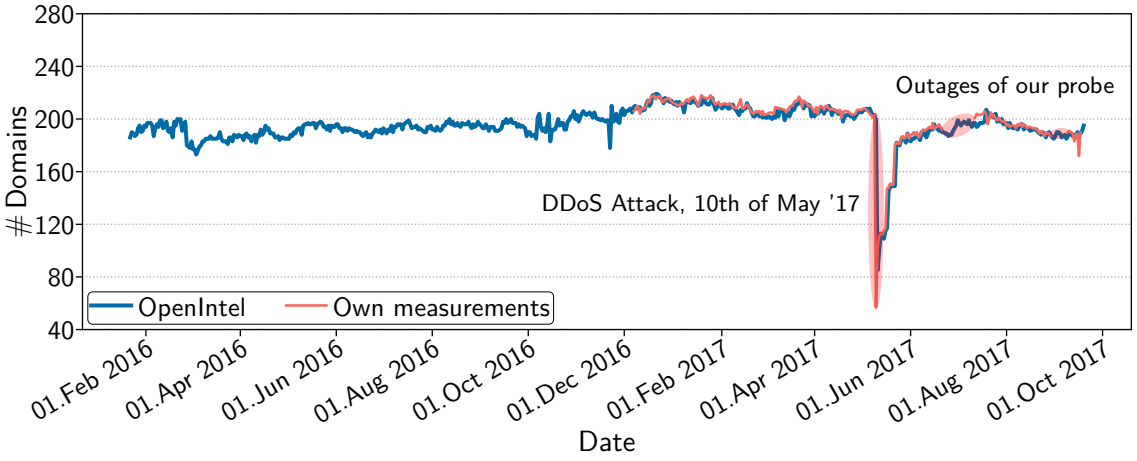


Figure 4.7 Domains utilizing Cedexis in the Alexa 1M over time. The drop in May '17 was caused by a DDoS Attack on Cedexis [CJ17]. Unfortunately, the measurement probe located at our chair experienced two outages. However, the overlap of both scans motivates the further use of the OpenIntel data set.

that optimizes for latency, and another for throughput. Conveniently, App IDs start at 1. Thus our approach is to simply enumerate customers by resolving all `2-01-(C_ID)-(App_ID).cdx.cedexis.net` domains. As customer and application ID each have four hexadecimal characters, we would need to probe > 2.5B (16^8) domains. To scale-down our DNS resolution, we only enumerate the first 256 application IDs for each customer, resulting in resolving roughly 16M domains.

Domain Lists

Second, we probe domain lists to study the usage of the enumerated CNAMEs in the wild and to discover application IDs beyond the enumerated 256 IDs. We thus resolve the A-record of `domain.tld` and `www.domain.tld` for all domains in the *i)* .com/.net (obtained by Verisign), *ii)* .org (obtained from PIR), *iii)* .fi (obtained from Ficora), *iv)* .se/.nu (obtained from IIS), *v)* .new gTLD zones (obtained from the Internet Corporation for Assigned Names and Numbers's (ICANN's) Centralized Zone Data Service), *vi)* obtained from our passive DNS probe, and *vii)* the Alexa Top 1M list. We additionally include the Cisco Umbrella Top 1M list [Cis16], which is based on the most frequent queries to OpenDNS resolvers and additionally contains subdomains, e.g., `images.domain.tld`. Depending on the size of the list, we perform daily or weekly resolutions for four weeks in August 2017 and extract all domains which have a CNAME pointer containing `*.cedexis.net`.

Customer List

We combine both data sets to a customer list that will form the basis for probing Cedexis globally in Section 4.2.3.

The list contains all customer application tuples, of which we discovered 84% in the enumeration step, and 11.2% in both the enumeration and in the domain lists, and

Type	Share	Type	Share	Type	Share
Business	17.7%	Unknown	8.1%	Social	1.6%
IT	12.1%	Goods	5.6%	CDN	1.6%
News	11.3%	Automotive	5.6%	Patents	0.8%
Gambling	11.3%	Advertising	3.2%	Banking	0.8%
Shopping	8.1%	Streaming	2.4%		
Gaming	8.1%	Television	1.6%		

(a) Classification

Service	Share
Web	62.7%
Unknown	15.6%
Assets	12.9%
Media	5.4%
API	2.3%
Bulk data	1.1%

(b) Services

Table 4.4 Cedexis customer information obtained from manual inspection of websites served for different customer IDs. Please note that customers may operate multiple services, e.g., multiple brands of one holding company.

4.8% solely in domain lists. The reasons for the latter are application IDs larger than 256, which were not part of our enumeration. Out of all customers, 55 (20) have only 1 (2) application(s) configured. We also observe one customer having 84 configured. By resolving the domain lists, we find 4609 (sub-)domains pointing to 16% of all discovered (customer, application) tuples. We did not hit the remaining 84% when resolving our domain lists. For 62.7% of all customer application IDs, we only find a single domain pointing to it. We find 31 (6) application IDs managing more than 10 (100) domains, respectively.

We show the popularity of Cedexis over time among Alexa-listed domains in Figure 4.7. We set up regular DNS resolutions in December 2016 and further show regular Alexa resolutions performed by OpenINTEL [vRJS+16] in the Netherlands for the same resource records. First, we observe that both data sets overlap, suggesting that both are suitable for monitoring. Minor fluctuations in the number of domains per day can mainly be attributed to fluctuations in the Alexa-listed domains [SHG+18]. Second, we observe an outage of Cedexis in May 2017, which was caused by a DDoS attack on their infrastructure [CJ17]. The outage motivated some customers to remove CNAME pointers to Cedexis in favor of pointing to operational CDNs instead, causing a drop of > 120 domains in Figure 4.7.

Customer Classification

We next classify the discovered customers to highlight the variety of Cedexis customer' profiles. To base this on an open classification scheme, we first tried to match customer domains against the Alexa Web Information Service API. However, Alexa classifications exist only for 17% of the queried domains, and some classifications do not reflect the Web pages' content. To obtain a broader picture, we instructed a single human classifier to visit each website and categorize it according to an evolving set of categories. We show the resulting categorized website content in Table 4.4(a). The table shows that a broad range of customers uses Cedexis. We further classify the used service in Table 4.4(b). The table shows that most customers use Cedexis for general Web content delivery. This list includes few but large bulk download services, e.g., `www.download.windowsupdate.com`. This observation is in contrast to, e.g., Conviva, which dedicates to video delivery.

Takeaway. *Several (large) Web services utilize Cedexis. Decisions taken by Cedexis have the potential to impact larger bulks of Internet traffic.*

4.2.3 A Global View on Cedexis

As Cedexis' customers can realize routing decisions based on (network) location, we next take a global view on its customers by using globally distributed active measurements.

Measurement Setup

We base our measurements on 35 PlanetLab nodes located in eight countries, six custom Raspberry Pi probes in six distinct German ISPs, and RIPE Atlas probes. We chose PlanetLab and custom Raspberry Pis in addition to RIPE Atlas since they enable us to deploy custom software to perform frequent DNS resolutions. As we do not include PlanetLab nodes located in Germany in our set, we refer to our deployed Raspberry Pis when mentioning *DE* in figures or plots. We selected only few PlanetLab nodes with high availability to repeatedly measure always from the same vantage points. For our measurement, we instruct the Planet Lab and our Raspberry Pi nodes to resolve each domain every 15 min and subsequently measure the latency to the resulting IP addresses. Moreover, we keep track of all CNAME redirections to CDNs that we observe throughout the measurement and also resolve these. This way, we learn the set of configured CDNs for every probed domain.

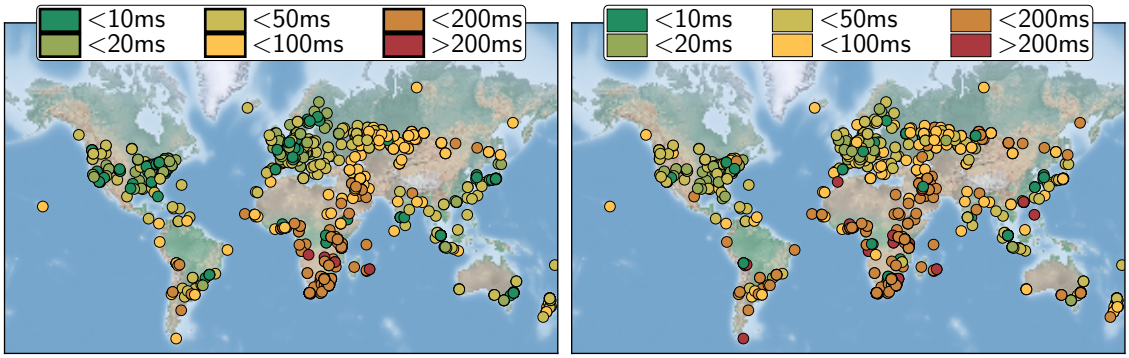
4.2.3.1 Infrastructure

We start our global perspective by taking a look at the infrastructure that is deployed and managed by Cedexis.

Authoritative DNS Deployment

Cedexis' core functionality is based on a distributed infrastructure of authoritative name servers managing `*.cedexis.net`. We find four servers configured in the DNS in our measurements and the `.net` zone file. We remark that a larger number exists which we found by enumerating their naming pattern. However, they currently appear to be unused, i.e., not included in the *.net* zone and are not discovered by our active DNS measurements.

To obtain a better understanding of its DNS infrastructure, we measure the ICMP echo (ping) latency to their authoritative name servers from ≈ 870 responsive (out of 1000 selected) RIPE Atlas probes. We repeated this measurement 30 times using the same set of probes and show the minimum RTT in Figure 4.8a. Based on these latency figures, we infer that Cedexis operates DNS servers located in North-America, Europe, and (probably) Asia and South America. By analyzing individual latencies and manual traceroutes per server-IP (not shown), we observe latencies of < 10 ms



(a) Minimum `ping` RTTs to the four Cedexis authoritative NS.

(b) Median DNS query times to resolve an A-record via Cedexis

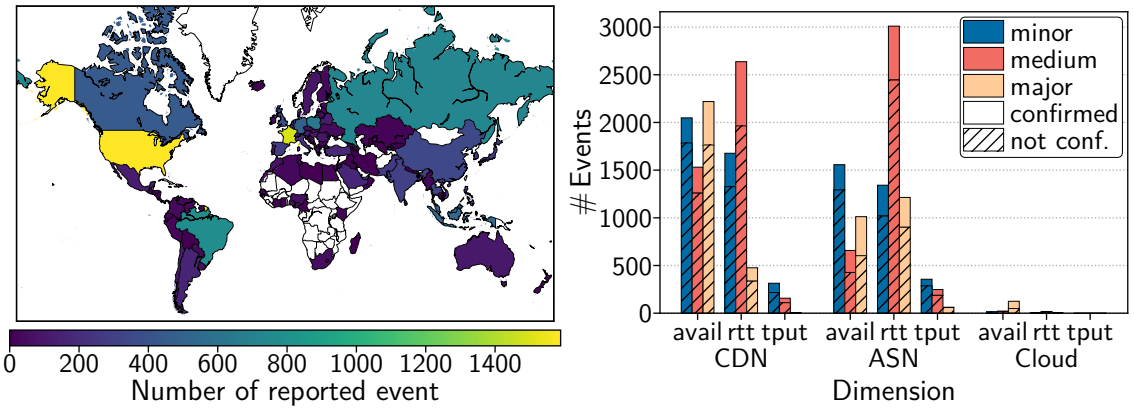
Figure 4.8 RTTs and DNS query times obtained from ≈ 870 responsive RIPE Atlas probes performing pings and DNS A-record requests.

from multiple regions (e.g., US-East, US-West, Europe, Hongkong) to the *same* DNS server IP address. Since these low latencies are lower than required by the speed of light between the respective regions, it suggests that the probed server IP addresses are served using *anycast* routing.

Since the additional indirection step through Cedexis contributes latency, we next measure the DNS resolution time of only their authoritative NSes. In this step, we resolve the domain of a Cedexis customer from all authoritative name servers from the same RIPE Atlas probes, again repeated 30 times. To limit the resolution to only involve the Cedexis NS, we chose a customer-domain which directly returns an A-record instead of redirecting to another CDN. This process is especially crucial if the lifetime of the DNS records is short (which we analyze in Section 4.2.3.2) and clients need to contact the Cedexis DNS over and over again. We show the median DNS query time in Figure 4.8b and observe that the DNS query times follow the previously measured ping latencies. Nevertheless, we observe few regions in which Cedexis appears to have suboptimal coverage as these regions involve high DNS resolution latencies, e.g., Latin America or Africa.

Radar Community Probes

Cedexis' customers can realize routing decisions that are based on active measurements of current service performance probed by visitors of Cedexis managed websites (Radar platform, see Section 4.2.2.1). Understanding this data is interesting since it can influence routing decisions. As the Radar data is not publicly available, we instead analyze live feeds of network events detected by Radar and published at `https://live.cedexis.com`. The events report three classes of metrics: *latency*, *throughput*, and *availability* for *ISP*, *CDN*, and *cloud* infrastructures. An event can be a latency in- or decrease, an outage, or a change in throughput. A CDN/cloud event is detected if visitors from five different ASes reported it. Likewise, an AS event is detected if it concerns five CDNs or clouds (see `live.cedexis.com`). Each event can be classified by severity into *minor*, *medium*, and *major*. Apart from



(a) Locations of confirmed events. (b) Event types, status, and severity.

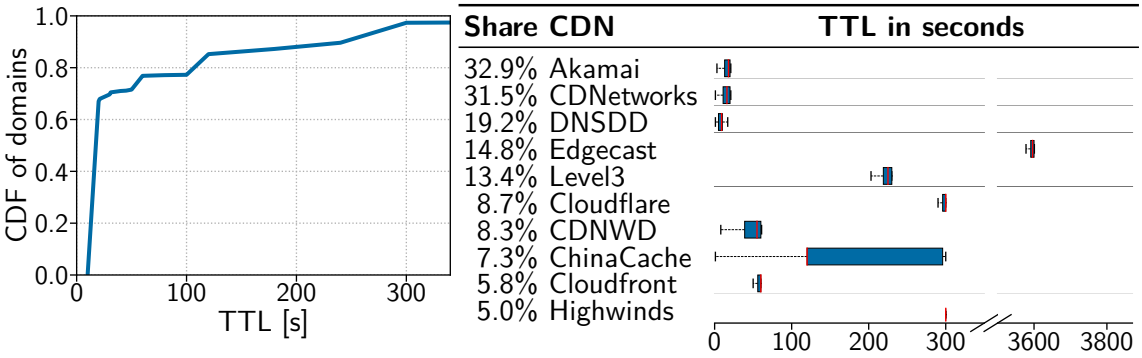
Figure 4.9 Cedexis events reported from 9th of October 2017 to 8th of January 2018.

being used in their decision-making process, this data allows monitoring the reported infrastructures. We thus monitored the data feed from 9th of October, 2017, to 8th of January, 2018.

Reported events further provide information about the *location* of visitors to these Cedexis-managed sites. Cedexis derives this information from the JavaScript measurement code that Cedexis' customers embed into their sites reporting performance figures to the Radar platform. While the number of events is likely uncorrelated with the number of website visitors, it still indicates the presence of a visitor from the reported AS or country. Therefore, we show the distribution of the number of events per-country in Figure 4.9a. We observe almost no events in Africa, suggesting that Cedexis' customers do not have a large user base in Africa, which also coincides with the suboptimal DNS deployment there. While we see events in almost every country, most events are reported in Central Europe, North America, Brazil, and Russia.

We next analyze the reported events by their type, shown in Figure 4.9b. The figure shows the number of events per event type categorized to availability (avail), latency (rtt), and throughput (tput) for CDNs, AS, and cloud providers. Every bar is divided into the number of confirmed and unconfirmed events. We observed that an event is marked as *confirmed* when it was reported for at least 9 min and the rolling variance of measurements from the last 5 h exceeds an event- and severity-level-specific threshold: e.g., a latency increase of 100% – 200% for a CDN is considered as minor, while an increase between 200% and 500% is considered as medium severe. We find most of the reported events to concern CDN, followed by ASes. The high number of major availability events concern CacheFly CDN outages during our measurement period.

Takeaway. *We observe visitors of Cedexis-managed sited from almost every country. However, its anycast DNS platform is likely based in the US, Europe, and Asia. Users in other countries can be subject to higher DNS query latencies.*



(a) CDF of customer-configured TTLs values from Cedexis to the serving CDN.

(b) TTLs of A-record for top 10 used CDNs. (Note the gap in the time scale to display Edgecast using anycast.)

Figure 4.10 DNS TTLs experienced among Cedexis-enabled domains. For (a) mappings from Cedexis to the subsequent entry, and (b) the CDNs used for the final delivery.

4.2.3.2 How Customers utilize Cedexis

To shine a light on how Cedexis is utilized, we take a look at configured DNS TTLs, how long the resolution takes as well as which CDNs are finally used to serve the content.

DNS TTL

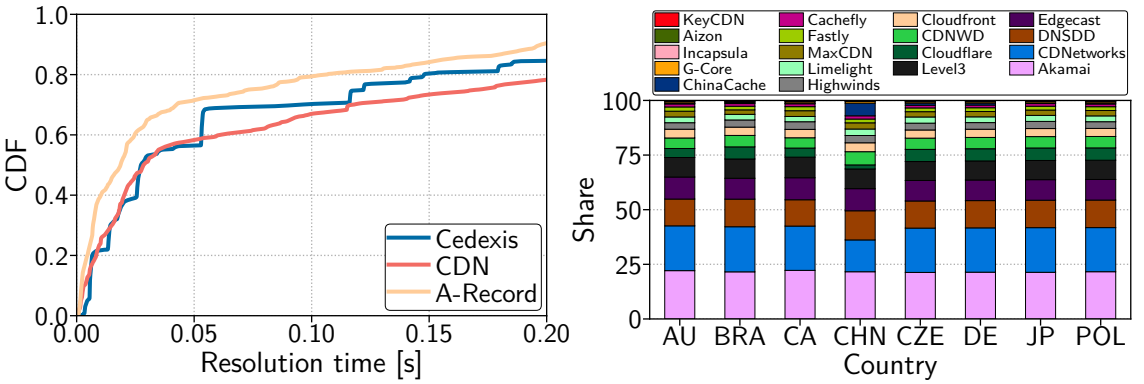
The DNS TTL defines the time a DNS resolver may cache a record and thus the timespan between Cedexis' balancing decisions. A small TTL allows more frequent switches at the cost of more frequent DNS queries to the Cedexis DNS infrastructure. This query latency can be significant, depending on the DNS resolver and NS location.

Figure 4.10a depicts the cumulative distribution function (CDF) of the TTLs for the validity of the CNAME-mappings from Cedexis to the subsequent entity (see 2nd CNAME in Figure 4.6) for all customer domains. We did not observe country-specific settings. Around 67% of all domains have configured a TTL of at most 20 s, indicating a rather short time scale enabling rapid reactions to changes. The next 30% are within 300 s, denoting an already moderate reaction time while around 3% have configured higher TTLs. Higher TTLs *can* hint to non-latency-based, but rather throughput or cost-based optimizations.

To compare these configurations to TTLs deployed by CDNs, we show the `A`-record TTLs for the top 10 CDNs in Figure 4.10b. To the right of every CDN, the figure shows the boxplot of TTLs observed for the `A`-records of all resolutions we performed. We see that the top 3 CDNs use a short TTL in the range of most Cedexis CNAMEs, whereas Edgecast has a lifetime of one hour (probably due to their use of anycast).

DNS Resolution Time

When employing Cedexis, an additional step in DNS resolution is required to enable CDN balancing. Figure 4.11a compares the latency for resolving (from our Planet



(a) Lookup latency. (b) Share of CDN choices per country.

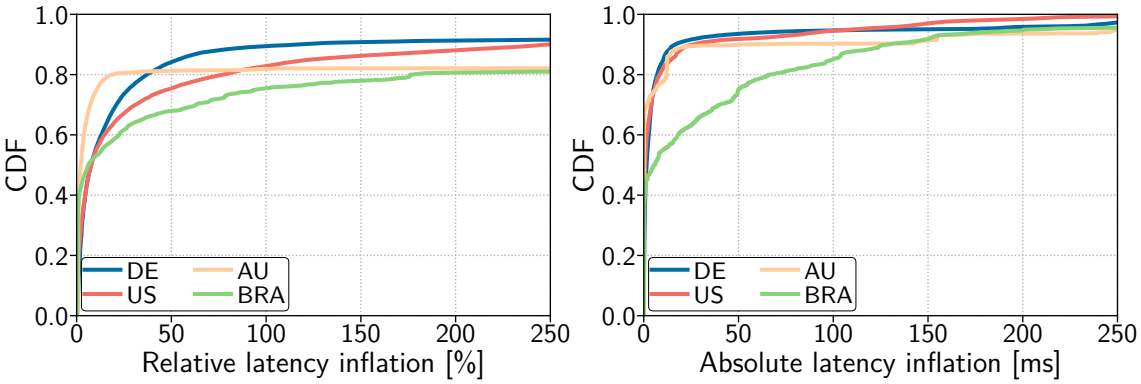
Figure 4.11 Lookup latency in DNS resolution and final CDN choices of Cedexis customers.

Lab sites) a mapping at Cedexis, in case of multi-staged CDNs a further CNAME redirect (CDN) and the final resolution of the A-record. We observe that Cedexis performs similarly to the other CDNs. However, while this hints at a good DNS deployment for our vantage points, it also means that using Cedexis inflates the latency of a DNS lookup. Given the on average short TTLs, users will often incur an additionally added latency when using Cedexis-enabled websites.

CDN Usage of Customers

We next check how many CDNs are being used by Cedexis' customers. Figure 4.11b illustrates the overall CDN selection frequency for every PlanetLab and Raspberry Pi node location over one month for all discovered Cedexis domains. We find that domains are usually only using one or two CDNs while there are a few that use more. This finding is consistent between geolocations. Only when looking at the how often which CDN is actually selected (note: Figure 4.10b did only show the share of domains using the CDN), we see a small geographic difference in China (CHN). Here ChinaCache is selected more often than in other geolocations. Nevertheless, apart from this, all geolocations behave similarly. This finding contrasts a finding on the Conviva network [MBM+16] showing a bias in which some CDNs are selected more often than others in specific countries. In summary, we do not observe that Cedexis' customers set country-specific routing decisions.

To extend our global view beyond the PlanetLab nodes, we next resolve the discovered Cedexis domains from open DNS resolvers obtained from `public-dns.info`. To avoid overloading (low-power) devices on user-premises (e.g., home routers), we exclude all resolvers whose DNS names indicate access lines (e.g., "pppoe", "dial-up", or "dsl"). We further only select resolvers with an availability >89%. In total, this leaves us with 1998 resolvers in 161 countries, out of which 67 *never* successfully responded. We resolve all Cedexis customer domains using all resolvers every two hours for four days. Subsequently, we group the reported results by continent and compare the top selected CDN. We observe that 66.9% always chose the same CDN in every continent. For the remaining, we observe disagreement, i.e., different CDN are chosen on each continent: 30.4% have two and 2.7% three CDNs present. We



(a) Relative difference to optimal choice. (b) Absolute difference to optimal choice.

Figure 4.12 Ping difference to the *fastest* CDN (RTT) when the choice was not optimal.

compare the complete CDN choices in countries of our PlanetLab nodes to their mapping results and observe similar distributions as in Figure 4.11b (not shown).

Takeaway. *Most Cedexis domains configure short TTLs to enable frequent switches. We observe that most domains indeed balance between a few CDNs. Switches pose a challenge to each CDN since traffic gets harder to predict.*

4.2.3.3 Latency Perspective

We next take a latency perspective on Cedexis' choices, i.e., comparing the latency of the chosen CDN to all configured CDNs for every customer domain. Thus, we measure the latency to every CDN IP by performing ICMP pings. We chose ICMP pings over more realistic HTTP requests since the pings do not generate accountable costs for the probed customers but remark that the ping latency can differ from actual HTTP response latencies. Still, the ping latency can be a reasonable performance and distance indicator for the different CDN caches.

Figure 4.12a shows the relative latency inflation for cases where Cedexis did not choose the latency-optimal CDN. We observe that around 50% of all resolutions are only marginally worse than the optimal choice regardless of the geographic location (the figure shows a selection). The curves then start to flatten, indicating increased latency inflation. We observe two groups, where around 10% (20%) of the choices exhibit a latency increase of over 250%. The observed increase can be in the range of few ms for nearby servers that would still deliver a good performance. Therefore, we show the absolute difference of all suboptimal decisions in Figure 4.12b. We observe that $\approx$ 50% of all decisions indeed only differ in a couple of milliseconds, indicating almost no noticeable difference. Apart from the nodes in Brazil and Japan (not shown), around 90% of all choices are even within a 20 ms difference.

We remark that it is difficult to assess the quality and correctness of CDN choices as we do not know the routing metrics that are employed by Cedexis' customers. Our measurements are motivated from the perspective of an end-user, who is interested in performance, not in potentially monetary business decisions.

Takeaway. *All available CDNs would deliver good latency figures in most tested cases, suggesting that the choice of CDN performed by Cedexis would not significantly impact end-user experience.*

4.2.4 Summary and Discussion

In this contribution, we empirically investigated how CPs can overcome strict policies enforced by Internet giants. Further, our contribution showed how policy-based routing is enabled regardless of an ossified and deprecated Internet core.

To this end, we presented a broad assessment of a Meta-CDN deployment, exemplified by dissecting Cedexis as a representative generic platform. By this, we enrich the existing literature describing the Meta-CDN concept with an empirical assessment of a large operator. We shed light on Cedexis' customers, technology, and performance. Cedexis DNS deployment, even though using anycast, appears to be focussing on Europe, North America and parts of Asia indicated by high latencies in other regions. We find customers to configure mostly low TTL values enabling fast reactions, and we indeed observe that most domains balance between few CDNs. By assessing ping latencies to all available CDNs, we observe that most available CDNs offer a good performance to our distributed probing platforms. However, we also find a range of suboptimal latency choices, which can indicate routing metrics other than latency.

The Meta-CDN operation challenges the demand modeling of the CDNs finally delivering the resources. The routing decisions implemented by the Meta-CDN customers affect the inbound traffic prediction at the CDN. This uncertainty especially amplifies when a significant fraction of the CDN's customers employ a Meta-CDN or similar concepts. In particular, routing decisions can be based on active measurements by the Meta-CDN — thus, bad performance can result in rerouting traffic and thus losing revenue. Studying Meta-CDNs and their consequences thus pose an attractive angle for future work.

4.3 Conclusion

Motivated by the large body of work on the transport layer circumventing ossified network designs, we first explored to what extent the network layer suffers from similar problems. To this end, we listened into ICMP backscatter produced by our measurements presented in Chapter 3 and discovered a substantial dataset that allows studying the ossification and deprecation. In summary, we found a plethora of outdated and misconfigured systems. This highlight the significant challenges when trying to innovate on the network layer, and as a consequence this reinforces the observation of little innovation on the network layer. Many of our findings show problems that were documented over two decades ago but are still present in today's Internet. We further discovered a significant degree of unreachability in IPv4 of which routing loops are the most severe reason which we even found at our upstream ISP.

Persuaded by these findings, we investigated Meta-CDNs that seem to be able to cope with these limitations and still innovate in the area of content routing. We dissected Cedexis as one of the largest generic Meta-CDNs and showed their operational model that does not at all rely on network layer innovation but instead solved this challenge with the help of an additional indirection layer within the DNS. Using globally distributed vantage points, we further analyzed how using Cedexis may impact their customers and found that they are not represented in all parts of the world equally, which could lead to suboptimal performance. Furthermore, we investigated how content owners are utilizing Cedexis and found that they generally configure the Meta-CDN to allow switching the serving CDN in short periods. By this, content owners regain control over these Internet giants that finally deliver their data and put them under pressure as end-user measurements and monetary decisions may now affect the profits of the utilized CDNs.

Driven by how Meta-CDNs innovate by pushing functionality onto the application layer, we next investigate how the evolution of new Web APIs on the application layer is currently abused in the Internet.

5

Abusing Innovation on the Application Layer

The two previous chapters have shown that ossification on the transport layer and in the Internet's core is significantly challenging the deployment and development of new technologies. Section 4.2 has shown that content providers (CPs) go up the stack to realize content routing with the help of DNS redirections. To this end, [RFC1925] jokingly admits that

> *(6) It is easier to move a problem around (for example, by moving the problem to a different part of the overall network architecture) than it is to solve it.*
> *(6a) (corollary). It is always possible to add another level of indirection.*
> — *Ross Callon [RFC1925]*

While there is a truth to this, protocols and applications on the application layer have, in contrast, been relatively easy to adopt and evolve. This ease is reflected in the swift evolution of the Web since the early 2000s moving from static to dynamic and interactive content. Figure 5.1 shows the standardization timeline of a selection of Web application programming interfaces (APIs) that are being standardized by the World Wide Web Consortium (W3C). The figure shows two things; first, the Web community is actively working on a plethora of new technologies and, second, the capabilities of the Web are rapidly expanding to new levels. To this end, a recent W3C community group proposal even grants native file system access to websites [Kru19], enabling, e.g., text editors that natively work with local files, i.e., a capability that has been reserved for non-Web apps. While this evolution is not without hurdles, e.g., Web browser compatibility has been a reoccurring issue for decades, it typically involves only the Web browser vendor and the website's hoster that uses the API, and both have a strong will to push the feature.

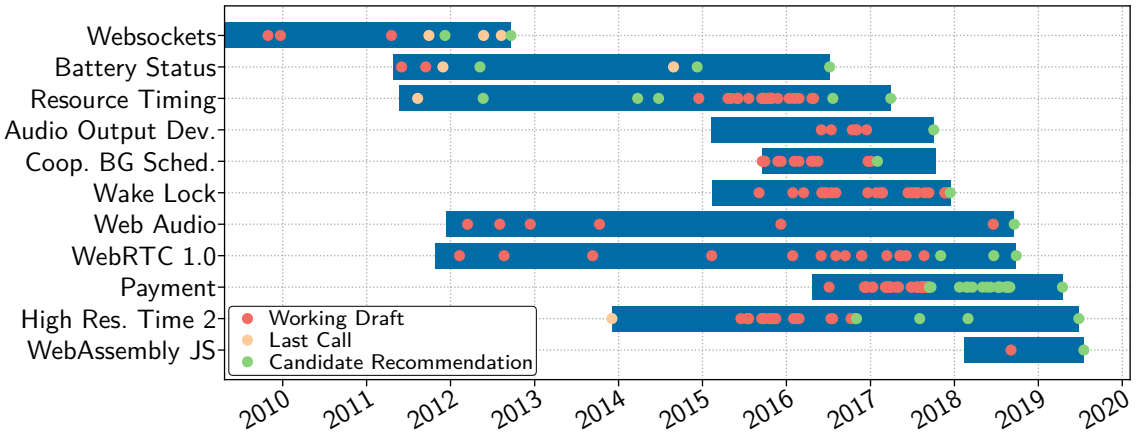


Figure 5.1 Standardization timeline (as of August 2019) of a selection of Web APIs documents from W3C which are candidates or proposed standards.

However, there is only limited research on the utilization of these new APIs in the Web at large. The largest body of work is in the context of Web browser fingerprinting (to reidentify user across website visits)[35]. Nonetheless, these works focus on the API's ability to fingerprint and do not take a look at the API use itself. In this chapter, we, therefore, want to investigate our last research question, *how are application-layer optimizations that are pushed forward by Internet giants used at large?*

Specifically, we want to focus on how this new technology is deployed and abused at large. Our approach to this is motivated by news and media reports about browser-based cryptocurrency mining, e.g., [Gua17a, War18]. Thus, our next contribution investigates the prevalence of browser-based mining in the Web and how application layer innovations are fueling it. For this, we have to overcome several research challenges.

Research Challenges

- **How can we inspect the dynamic nature of websites on a large scale?**

  Today, websites have become increasingly complex and are often heavily modified after initial loading with the help of JavaScript. These scripts can either load additional scripts or resources or modify the website's structure. These dynamics make it increasingly difficult to analyze the Web as merely downloading the HTML is not sufficient. We thus need to find a way to execute the dynamic nature of today's Web on a large scale such that our analyses can cover large parts of the Web.

- **Privacy-preserving blockchains make it seemingly impossible to identify users.**

  Modern blockchain-based cryptocurrencies are designed with privacy in mind. For example, Monero uses ring signatures to hide a single user in a larger

[35] A survey can be found in [LBB+19].

group and stealth addresses for receiver privacy increasing the anonymity set. To study the prevalence of browser-based mining, we nevertheless want to find a way to associate mined blocks with a miner, which is possible in non-privacy-preserving blockchains by looking at the transaction that pays the miner.

5.1 Browser-based Cryptocurrency Mining

The Web economy has traditionally used advertisements as a means to monetize services that are otherwise offered free of charge. This business model relies on the implicit agreement between CPs and users where viewing ads is the price for the "free" content. This traditional approach has very recently been complemented by a new monetizing model in which the computational resources of website visitors are used to mine cryptocurrencies to generate revenue for the website operators (browser-based mining).

Mining is the method of producing new blocks in blockchain systems, most prominently cryptocurrencies such as Bitcoin. It requires miners to solve a computationally expensive puzzle to cryptographically link a new block to the previous block in the blockchain. The difficulty to solve this puzzle depends on the combined computing power of all users — depending on the difficulty, an individual requires powerful machines to increase the probability of mining a block (e.g., GPUs, FPGAs, or even ASICs). The system provides an incentive for contributing this computing power by awarding miners with currency for every mined block. This monetary reward has rendered crypto mining a business — browser-based mining extends this business to monetize the Web.

Not all cryptocurrencies are equally suited for browser-based mining. The hardware imbalance in Bitcoin (resulting in a high mining difficulty) renders its in-browser mining inefficient and motivates the use of, e.g., Monero as an alternative currency that can be efficiently mined on CPUs and thus browsers. Given its design, websites have adopted Monero (e.g., The Piratebay or a video streaming service with subsequent media exposure [Gua17a, War18]) and even botnets utilize it to mine on millions of compromised hosts [Pro18]. To ease browser mining, APIs [Coi18, Cry18] exist, e.g., for in-game financing [DeV17], link forwarding [Rüt18b], captchas, during video streaming [Gua17b] or even as an entry fee for parties [OM18]. Our work identifies Coinhive [Coi18] as a widely used service which provides a framework for embedding a Monero miner into a website. While these frameworks enable mining without the users' knowledge (cryptojacking), other services (Authedmine) actively ask users for their consent to do so as an alternative to displaying ads. Besides media reports, little is known about the ubiquity and use of browser-based mining. We find that this mining has only become efficiently possible with the help of recent developments for Web APIs. Our findings show that WebAssembly (Wasm) and Websockets are the two driving technologies which enable browser-based mining as, primarily, Wasm offers an immense performance advantage over pure JavaScript to perform the computationally expensive mining.

Given these new possibilities, we provide a first in-depth study of the *prevalence* and *economics* of browser-based mining as a new Web business model. We base this perspective on crawls of 137M .com/.net/.org domains and the Alexa Top 1M list to first identify sites using browser-based mining, enabling us to create a new fingerprinting method to identify mining code. Our analysis shows that 96% of all Wasm code in our data is mining code. Thus, when mining is done without the user's consent, we find that Wasm is currently predominantly abused. Second, we dissect the short link service of the largest Web-mining stakeholder Coinhive and screen their market power and profits. Our contributions are:

- We investigate the prevalence of browser-mining in the three largest top-level domains (TLDs) and the Alexa Top 1M, i.e., at over 138M domains.
- We present a new Wasm-based fingerprinting method showing the inadequate capabilities of blocklists to detect mining.
- Moreover, we identify the largest browser-based mining-provider Coinhive and dissect their link-forwarding service.
- We present a novel methodology enabling us to associate blocks in a privacy-preserving blockchain to a mining pool.
- By applying our methodology, we screen Coinhive and show that they contribute 1.18% of the blocks in the Monero blockchain mining Moneros worth 150 000 USD per month (as of August 2018).

Structure. Before taking a look at the extent of browser-based mining, we first establish the basics of blockchains and the mining process in Section 5.1.1. Then, Section 5.1.2 measures the prevalence of browser-mining. Subsequently, we study the practices, user base, and economics of Coinhive as the largest browser-mining API provider in Section 5.1.3. Lastly, Section 5.1.6 discusses related work, and Section 5.1.7 concludes this contribution.

5.1.1 Excursus: Browser-based Mining 101

Blockchain-based cryptocurrencies build on the principle of embedding financial transactions in a public, tamper-proof series of blocks. The blockchain evolves by continually appending new blocks storing the currently pending transactions of the system; the block generation is called *mining*. Miners solve a crypto puzzle as a proof of work (PoW) whose *difficulty* is dynamically adjusted to produce new blocks at a constant *block rate* guaranteeing predictability and tamper resistance. Consequently, when more miners compete for finding blocks, the difficulty rises such that the system still meets the predetermined block rate. When the PoW meets the difficulty, it links the newly mined block (containing new transactions) to the previous one rewarding the miner with *currency* in exchange for the contributed computing power.

The recent hype around cryptocurrencies has led to substantial increases in difficulty resulting in the need for faster hardware to mine blocks profitably concerning energy costs. To increase the chance of earning currency, miners seek to increase their available computing power. Graphics Processing Units (GPUs), field-programmable gate arrays (FPGAs), or even specialized application-specific integrated circuits (ASICs) currently serve this quest for speed. One can host substantial quantities of

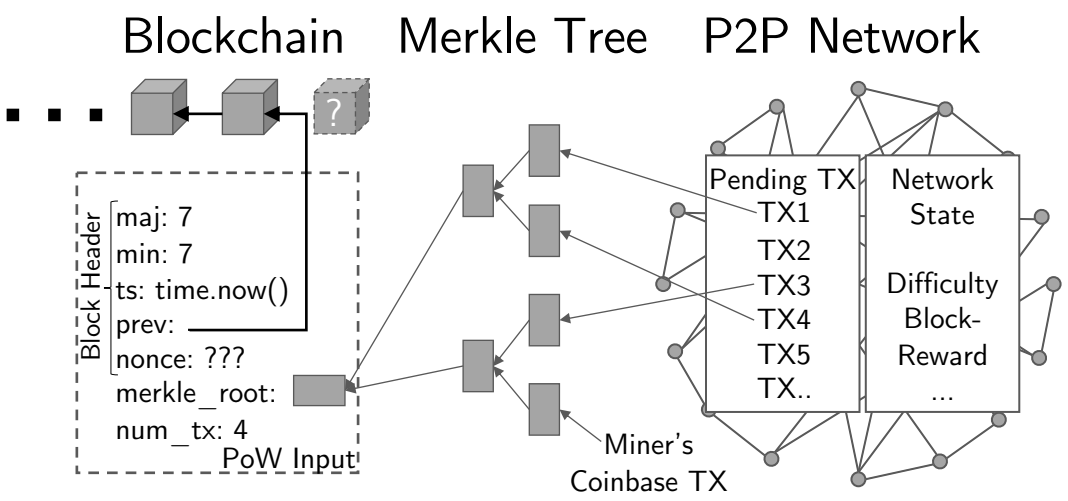


Figure 5.2 Monero blockchain and PoW mining input.

mining hardware in dedicated data centers. Another way is to bundle the computing power of multiple miners in *mining pools* that share the earned revenue for the newly mined block.

Browser-based Mining

Utilizing the computation power of website visitors provides yet another mean of increasing the mining power. By embedding mining code into websites, a miner can make use of the visitor's central processing unit (CPU) resources during the visit. The website operator thereby saves energy costs and mining hardware investments. Thus, Web-based mining is an alternative revenue-generating model to monetize websites and services. However, hidden mining or without user consent (i.e., cryptojacking) poses a significant challenge, and it is a known attack vector (Section 5.1.6). While browser miners for Bitcoin exist (e.g., jsMiner from 2011 [Whi11]), the performance imbalance between CPUs, GPUs, and ASICs poses an insurmountable challenge for Bitcoin browser mining. Consequently, browser-based mining requires cryptocurrencies with PoW functions that are only efficiently computable on CPUs.

Monero

Launched in 2014, Monero [Mon18] (see Figure 5.2) is a privacy-preserving cryptocurrency whose PoW is designed to be ASIC resistant (memory intensive and periodically redesigned) enabling CPUs and thus browser-based mining. Specifically, it uses the Cryptonight hash function [SJN+13] in its PoW to mine a new block with an average block rate of two minutes. Figure 5.2 shows the PoW inputs; in Monero, a miner constructs a Merkle tree of the transactions that are to be included in the new block, requiring at least the *Coinbase* transaction paying the block reward to the miner. A node in this tree is the hash of its two children with the individual transactions' hashes as the tree's leaves. Including the tree's root links the transactions to the PoW and the final block. Now the miner's goal is to find a nonce such that the PoW

output (a hash) meets the global difficulty (here, literally the product of the hash multiplied by the difficulty must be smaller than 2^{256}). Thus, a miner needs to draw a new nonce and recompute the hash until it satisfies this goal. The network can easily verify that the proof holds through a single round of hashing. By including the block in the blockchain, it awards the miner with the block reward expressed through the Coinbase transaction. When using mining pools, the pool pushes jobs (containing the PoW input) asking the miners participating in the pool to find a nonce that satisfies a lower difficulty than that of the entire network. When the miner solves the puzzle subject to this lower difficulty, then the pool awards the miner with a share of the final block reward, and if, by chance, the puzzle's solution also meets the network's actual difficulty, the pool mined a block.

5.1.2 Prevalence of Browser Mining

We start our analysis of browser-based mining by investigating its prevalence in the Web. Thus, we visit landing pages of a large body of domains and identify the presence of mining code using two approaches. Initially, we use a light-weight approach to download website landing pages via Transport Layer Security (TLS) across several datasets, i.e., .com (∼116M), .net (∼12M), .org (∼9M), and Alexa Top 1M (∼950K), and match their Hypertext Markup Language (HTML) body against a public filter list (Section 5.1.2.1). Subsequently, we instruct a Chrome browser to visit a subset of these domains to execute the Web page code and thereby monitor Websocket interactions and Wasm code as prevailing techniques for browser-based mining (Section 5.1.2.2). We obtain our datasets through Domain Name System (DNS) resolutions similar to our previous studies in Chapter 3 and Chapter 4 from zone files available at Verisign [Ver19] (.net/.com) and PIR [Pub19] (.org).

5.1.2.1 NoCoin List

We visit every domain, prefixed with `www.`, via TLS and download the first 256 kB of the domains' landing pages using `zgrab`. 256 kB offers a good tradeoff between capturing most content (i.e., scripts are often located in the head of the document) and having a point where to stop downloading when pages do not stop sending data. We then extract all JavaScript tags using `lxml` to apply the NoCoin filter list [Hos18]. This list contains regular expressions to detect and subsequently block mining code using common ad blockers. Figure 5.3 shows the number of domains with hits to NoCoin filter rules on the top x-axis. Relative to the number of domains, the bars on the y-axis show the relative share of the top 5 mining scripts (multiple per website possible). We find the prevalence of mining websites to be rather low. In comparison, (popular) Alexa-listed domains have the largest share (up to 0.07%). This unevenness seems likely since mining is most profitable with websites having many visitors. Looking at the miners, we find Coinhive, a Monero-based miner to be most prevalent (used by $>75\%$ of the mining sites). Notably, Authedmine, a variant of Coinhive asking for explicit user consent to mine and *wp-monero* a WordPress plugin follows but at much lower shares. We find other miners with smaller shares,

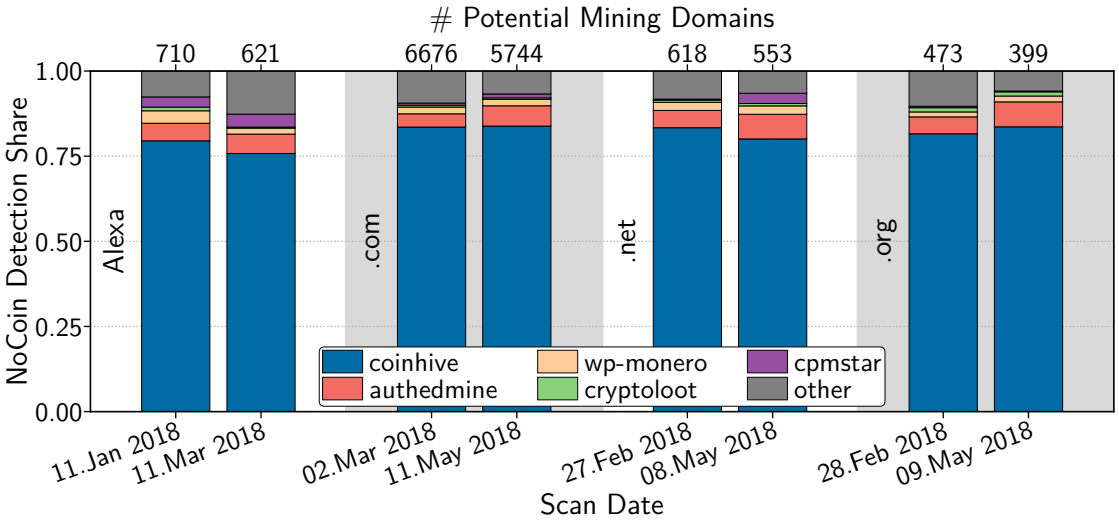


Figure 5.3 NoCoin detected miners on the Alexa Top 1M and .com/.net/.org domains.

e.g., Cryptoloot a Coinhive clone. By manually inspecting a random subset, we find false positives, e.g., *cpmstar* is a gaming ad-network that we could not verify to contain mining code. For the once-popular jsMiner [Whi11], we find only 31 instances in all datasets combined. **Takeaway.** *We observe a low prevalence of mining in landing pages according to the NoCoin list. Most miners are Monero-based with Coinhive having the largest share (*$> 75\%$*).*

5.1.2.2 Chrome

We complement the NoCoin analysis by broadly investigating patterns of mining behavior when further *executing* the pages. Accounting for the dynamics enables us to find mining domains beyond NoCoin-listed pattern. Through manual miner code inspection, we find that the majority of JavaScript miners utilizes Wasm for efficient PoW calculation. WebAssembly [Web18] is a binary instruction format — featured in all recent (mobile) browsers — that enables to compile, e.g., C-code to Wasm for efficient execution within the browser. Further, the communication to the backend servers providing the PoW input in a mining pool often uses Websockets. Both technologies have been pushed by large Internet giants in recent years to provide new means to utilize the Web (see Figure 5.1). To detect these, we instrument a stock Chrome Web browser using the Chrome Dev Protocol [Chr18a] to capture all Websocket communication and to dump all detected Wasm code. One of the major challenges is to decide when a (dynamic) page has fully loaded its content when trying to inspect a large body of websites in a feasible amount of time. Notably, since the website's HTML must not necessarily directly include the mining code. Often, scripts dynamically load the mining code after the rest of the page has loaded. This practice has two reasons; first, it offers means to obfuscate the existence of the mining code and, second, loading the mining code will not prolong the loading of the website's (actual) content. To decide when a page is fully loaded, we wait for the page's load event which the browser typically calls when it finished rendering the body of the website however scripts may still alter the Document Object Model

(DOM) and load further resources. Thus, we also set a 2 s timer on every DOM change but wait no longer than an additional 5 s before we mark the page as loaded completely. This method offers an upper limit of the waiting time (and limits the time we might actually mine) while being flexible to dynamic changes to the website. In case of no load event, we wait no longer than 15 s to mark the website as timed out. We further save the first 65 kB of the final HTML enabling us to compare with the NoCoin list used previously. As this measurement is more time consuming, we restrict our scope to the .org zone and the Alexa 1M. We prefix every domain with `http://www.`, allowing Chrome to follow redirects to the secured variant if necessary. Thus, in contrast to our previous TLS-only measurement, we also analyze non-Hypertext Transfer Protocol Secure (HTTPS) websites.

Miner Classification

To now detect mining code, we build a database for Wasm classification. To this end, similar to antivirus software, we create a characteristic signature of the Wasm code that we find. We build this signature by first combining the different Wasm functions to a big blob (in strict order) and then apply a SHA256 hash function to this blob to derive a signature. This hashing allows us to boil down the exact same code to the same signature, however, is undoubtedly prone to minimal modifications of the code. Through manual inspection of the Wasm, we build up a database of ∼160 different assemblies (often versions of the conceptually same miner) that we found and categorized them, e.g., through their Websocket communication backend or by some other distinguishing feature that we found in the code. Such features, e.g., comprises the number of XOR, shift or load operations which we found to be quite distinctive or function names hinting at the hash function itself. We also experimented with using only these features for Wasm classification, so without building an explicit database, that would allow being less prone to modifications. This feature-based classification was challenged by the low availability of Wasm analysis tools and frameworks. However, we found that a single feature is not enough to accurately classify the code and that a combination is required. Given that our 160 signatures proved to cover our data, we refrained from further developing the classifier but remark the interesting angle for future work that could benefit from machine learning. We believe that such a Wasm feature analysis might become a useful future browser feature to block certain (unwanted) functionality, especially in light of fingerprinting and user privacy. However, given the deep embedding of Wasm in the browser's JavaScript engine, this is likely not something that could be offered by a thrid-party browser extension.

Measurement Results

Table 5.1 summarizes our findings for the Alexa 1M and the .org TLD from measurements in the first two weeks of May 2018. We observe that most Wasm code contains mining functionality, and most miners are again Coinhive. To put the Chrome-based approach in perspective to the NoCoin list, we apply the NoCoin blocklist to the HTML saved by Chrome, i.e., after the execution of any JavaScript.

	Alexa Class.	Count	.org Class.	Count
1	coinhive	311	coinhive	711
2	skencituer	123	cryptoloot	183
3	cryptoloot	103	web.stati.bid	120
4	UnknownWSS	56	freecontent.date	108
5	notgiven688	46	notgiven688	92
Total	WebAssembly	796	WebAssembly	1491

Table 5.1 Top 5 (∼80%) WebAssembly signatures. Most WebAssembly codes are miners (∼96%), dominated by Coinhive.

	NoCoin Hits	having Wasm Miner	Wasm Hits	blocked by NoCoin	missed by NoCoin
Alexa	993	129	737	129	608 (82%)
.org	978	450	1372	450	922 (67%)

Table 5.2 Miners on Chrome data (incl. non-TLS) found through NoCoin and by our WebAssembly signatures.

Table 5.2 shows the number of miners detected by the NoCoin list and the fraction of mining Wasm on this part as well as the total number of websites classified through our Miner Wasm signature database and the subset of websites that were detected by the NoCoin list. We observe that NoCoin classifies many websites as miners, of which only a fraction really embeds mining Wasm code. This attribution indicates false positives which we verified through random inspections. If we take a look at the websites for which we found Wasm mining signatures, again, the NoCoin list only classifies a fraction of these as having a miner — false negatives.

Website Classification

We use the Symantec RuleSpace[36] [Sym18] engine to categorize the mining websites. Table 5.3 shows the top 5 categories to which RuleSpace assigned the websites for the NoCoin list matches as well as our signature-based approach. We observe a diverse set of categories and that RuleSpace can classify a larger body of Alexa domains than .org domains. Interestingly, the categories for NoCoin and our approach differ, especially the top category shows a large mismatch, i.e., Gaming vs. Pornography and Gaming vs. Religion. This mismatch could be caused by the gaming ad-network mentioned earlier, but in general, it shows that using only the NoCoin list paints a false picture of who is actually embedding this code.

Takeaway. *Miners are already embedded on websites today. Simple blocklists are ineffective to detect them all, and our signature-based approach can detect sites beyond the NoCoin blocklist. Still, Coinhive is the most used mining service.*

[36] Used in Symantec products to classify websites.

	Alexa				.org			
	NoCoin		Signature		NoCoin		Signature	
1	Gaming	19%	Pornogr.	19%	Gaming	29%	Religion	9%
2	Edu. Site	9%	Tech.	8%	Business	8%	Business	8%
3	Shopping	8%	Filesharing	8%	Edu. Site	6%	Edu. Site	8%
4	Pornogr.	7%	Edu. Site	5%	Pornogr.	5%	Health Site	7%
5	Tech.	6%	Ent. & Music	5%	Shopping	4%	Tech.	6%
Categorized		79%		74%		54%		42%

Table 5.3 Top 5 categories, according to Symantec RuleSpace.

5.1.3 The Coinhive Service

Coinhive provides the most widespread miner and mining API (see Section 5.1.2) and advertises its services with the slogan

> *Monetize Your Business With Your Users' CPU Power*
> — *Coinhive [Coi18]*

Their services are built on providing a highly optimized Monero JavaScript and Wasm miner to be embedded in websites. In turn, Coinhive keeps 30% of the mined reward. Apart from offering this API, Coinhive offers, e.g., a Captcha service and a short link forwarding service, which is the subject of our first analysis. Our tools on which the following analysis is based are available at [Rüt18a].

Regardless of the actual service, the process works as follows: *i)* A Coinhive user (e.g., a website owner) is assigned a unique token that is included in the API calls which is used to associate the mined shares. *ii)* Upon a website visit, the miner is loaded and it connects to the Coinhive mining pool and authorizes with the user's token to receive input for hashing. *iii)* Once a website visitor finds a valid hash, it is committed to the Coinhive pool. *iv)* Eventually, Coinhive pays its users 70% of the block reward and keeps the remaining 30%.

5.1.4 Short Link Forwarding Service

To begin analyzing Coinhive, we focus on its short link forwarding service, which is similar to a standard short link service (e.g., bit.ly) but additionally requires to compute a configurable number of hashes before resolving the link. When a user visits an uniform resource locator (URL) of the service, she only sees a progress bar indicating the share of hashes that her browser has already computed. When all locally computed hashes have been committed to the service (i.e., the progress bar is full), the service will return the original URL and instructs the browser to redirect the user to it. This link redirection monetization is comparable to short link services delaying the redirection while serving advertisements and paying the link creator a commission as analyzed by Nikiforakis et al. [NMS+14]. With Coinhive, the creator

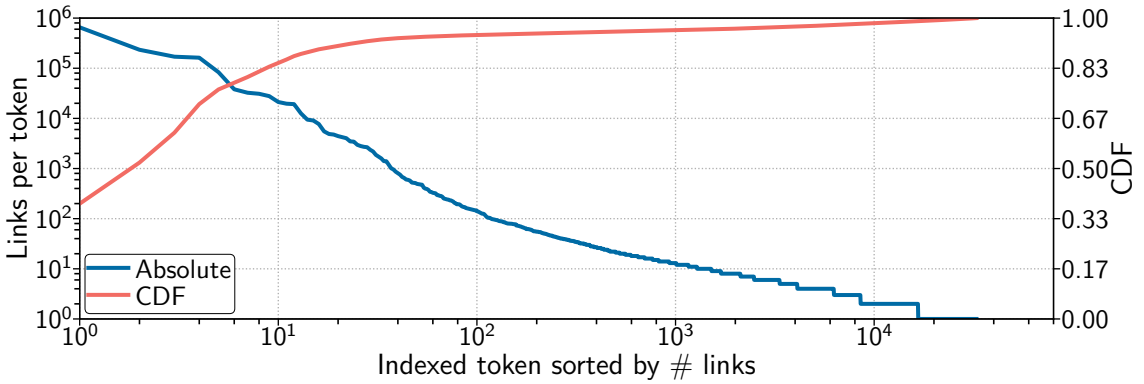


Figure 5.4 The number of links per token (users) is heavily biased towards a small number of users.

of the short link receives a share of the block reward that is mined by the users visiting the short links.

Their short links follow a simple structure, identified by an alphanumeric ID trailing the end of each link: `https://cnhv.co/[a-z0-9]`. We observed that new links are assigned increasing IDs, which enables us to enumerate the link address space (a common privacy failure with link forwarding services). As of February 2018, up to four characters are used, resulting in a total of 1,709,203 active short links. We visit all links and gather the Coinhive redirection HTML document to collect, *i)* the link creator's token — used to associate performed hashes to the link creator — as well as, *ii)* the number of hash computations required by the link creator to resolve the link. Even though a single user could own multiple tokens, we will regard users and tokens as synonymous here.

Without actually computing hashes, we can already look at *i)* the distribution of short links per Coinhive users as well as *ii)* the required number of hashes to resolve the links. Figure 5.4 shows the distribution of short links per user. We observe a power-law which highlights the existence of a few heavy users that created a large number of links. In fact, only a single user contributes 1/3 of all links, and only ten users create roughly 85% of all links. Of course, a single user could use multiple tokens; however, this would only emphasize our current observations.

To truly resolve the link, the user needs to compute the required number of hashes set by the link creator. Figure 5.5 shows the distribution of this link resolution difficulty in the number of required hash computations. The dark blue portion of the cumulative distribution function (CDF) depicts all observed links, while the light red portion removes the previously observed bias by heavy users by counting a required # hashes only once per user; i.e., 1000 links from the same user with the same number of required hash computations are only counted once instead of 1000 times as in the dark blue dataset. To provide a perspective on the time it takes to resolve a short link, we show the duration to compute the required number of Cryptonight hashes in a Chrome browser with a commodity[37] laptop on the top x-axis. We observe that our laptop can resolve the majority of links in less than

[37] 2013 Macbook Pro 2.8 GHz Intel Core i7: 20 h/s with 4 threads.

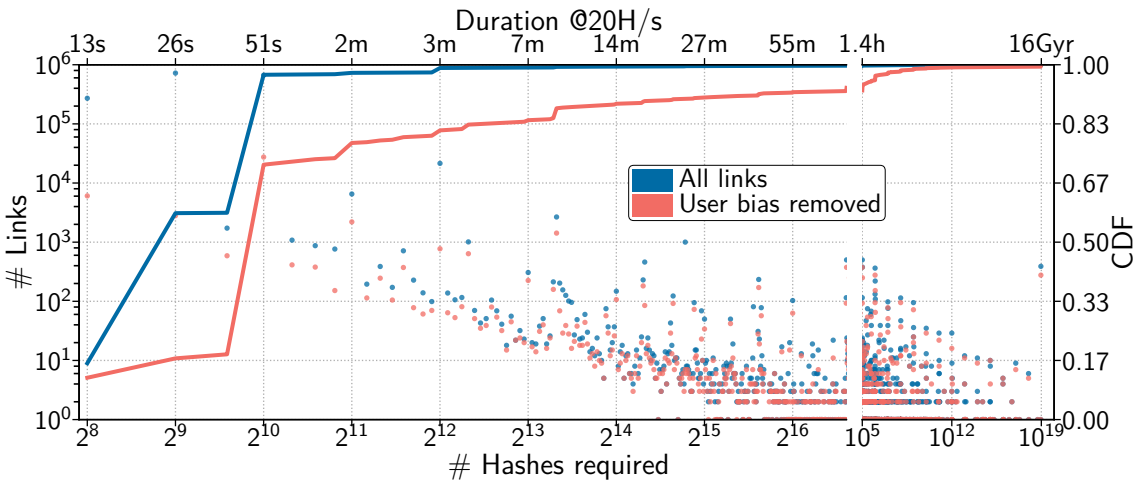


Figure 5.5 Required number of hashes and their frequency of occurrence as well as the time it takes to compute these hashes. Please note the sliced and different-scaled x-axis.

51 s (1024 hashes). The heavy user bias is most prominent at 512 hashes, still, when removing the user-bias over 2/3 of the links of all users can be solved with at most 1024 hashes in below one minute. To our surprise, many links require a longer time to resolve; we find many different users and over hundreds of short links that set the maximum of 10^{19} hashes, which takes several billion years to resolve. While the user's willingness to wait certainly depends on the content that is supposed to be behind a short link, high values suggest either no desire to have them ever resolved or misconfigurations (e.g., short link creators are not aware of the actual duration).

Link Destinations

To understand the kinds of links that the short link service is used for, we resolve all links which require less than 10k hashes from the unbiased dataset (covering 85% of this dataset, see the light red CDF in Figure 5.5). Additionally, we resolve a random sample of 1000 links for each of the top 10 Coinhive users. To efficiently resolve the short links without a Web browser, we replicate the working principle of the Web miner in a non-Web implementation that can resolve multiple short links in parallel making use of the official optimized Monero hash code. We found that Coinhive alters the block header contained in the PoW inputs before sending them to the users which the Web miner reverts deep within its WebAssembly[38]. This modification appears to be a countermeasure to prevent using the Coinhive Web miner outside of the Coinhive environment, e.g., in custom mining pools. We had to roughly compute 61.5M hashes which we were able to do in little less than two days on a capable server machine.

Top 10 User

We first investigate a random sample from all short links of the top 10 users (1000 links each) representing 80% of the link targets. Table 5.4 shows the classification

[38] A simple XOR with a fixed value at a fixed offset.

Domain	Category	Freq.	Domain	Category	Freq.
youtu.be	Ent. & Music	20%	ftbucket.info	Msg. Board	9.9%
zippyshare.com	Filesharing	10%	getcoinfree.com	Finance	9.2%
icerbox.com	Filesharing	10%	ul.to	Filesharing	4.2%
hq-mirror.de	Ent. & Music	10%	share-online.biz	Filesharing	2.9%
andyspeed racing.com	Automotive	10%	oboom.com	Filesharing	2.8%

Table 5.4 Top 10 domains present in 89% of all samples from the top 10 short link creators.

Category	Count	Category	Count
Tech. & Telecomm.	1522	Shopping	572
Gaming	737	Finance and Investing	502
Dynamic Site	727	Ent. & Music	313
Business	578	Educational Site	305
Pornography	577	Hosting	298

Table 5.5 Top 10 categories of the unbiased dataset $<$10K hashes.

for the top 10 domains (accounting for roughly 89% of all sampled URLs) that we extracted from the destination URL. We again utilize the RuleSpace categories to classify those ten domains manually. As the table shows, most links point to streaming and filesharing services.

Top Categories

We employ the RuleSpace engine to further classify the unbiased dataset into categories. One URL can have multiple categories; therefore, a single URL can contribute to different categories. For roughly 1/3 of the URLs RuleSpace has no classification, for the remainder, Table 5.5 lists the top 10 categories and how often a URL falls into each category. We observe that sites fall into a diverse set of categories, unlike the top 10 users for which filesharing and streaming were the dominant categories (Table 5.4).

Takeaway. *Ten users dominate Coinhive's link forwarding service. The links mostly redirect to streaming videos and filesharing sites. We find that most short links can be resolved within minutes; however, some links require millions of hashes to be computed, which is infeasible.*

5.1.5 Estimating the Network Size

While we find many websites to use Coinhive (see Section 5.1.2), it remains unclear how many users visit these sites. Thus, the mining power and possible payouts are unknown. To understand the available mining power and thereby the users of Coinhive, we need to identify which blocks in the Monero blockchain were mined through Coinhive.

Methodology

When the Coinhive network mines a block, one of the clients must have found a nonce that satisfies the PoW difficulty. Then, a new block can be mounted into the blockchain which contains the block header that is also part of the PoW input, as well as all the transactions that have implicitly been included in the PoW input through the Merkle tree root (see Figure 5.2). Thus, if we find the PoW input for which a suitable nonce was found, we can investigate the blockchain and look at the block that succeeds the block referenced in the PoW. If the transactions in that block form a Merkle tree whose root is equal to that in the PoW input, we can be sure that the PoW input was the one that was used to mine the block. This root thus uniquely identifies the origin as each block contains the Coinbase transaction (first leaf of the Merkle tree) which is used to pay the block rewards to the miner (i.e., Coinhive). Thus we could never by accident see a Merkle tree root of another miner in the PoW input.

We investigate the PoW inputs that are delegated by Coinhive to its miners by connecting to one of their mining pools and request a new PoW input every 500 ms. As the network finds a new block on average every two minutes, we cluster the PoW inputs by the pointer to the previous (at time of reception, most recent) block. We found that we never obtain more than eight different PoW inputs (even though more exist theoretically by permuting the transactions in the Merkle tree). Coinhive currently operates 32 mining endpoints (which can be gathered from the JavaScript or by enumerating the domain name), when we connect to all of them and repeat the process, we observe at most 128 different PoW inputs per block. While this suggests that there are two endpoints per backend system, it also puts us into the position to investigate each of the 128 PoW inputs.

For each PoW input, we thus compare the Merkle tree root against the Merkle tree root that is saved in the public blockchain. To find the right block to compare to, we look at the block header from the PoW input (see Figure 5.2) and obtain the ID of the previous block to which this PoW input's block header is supposed to be the successor. We then take a look at the blockchain and obtain the block that is truly succeeding this previous block, i.e., the block that was ultimately mined. If this block's Merkle tree root is equal to that from the PoW input, this block must have been mined through the Coinhive mining pools as the Merkle tree root is unique for each miner.

Measurements

We have been requesting new PoW inputs for four weeks, and we are thus able to confidently estimate a *lower bound*[39] on the blocks mined through Coinhive. Figure 5.6 shows a blue block for every Coinhive-mined block as well as the total number of blocks on that day. As finding blocks correlate with users mining through Coinhive, we were interested to see if blocks are found at certain times, which could

[39]There is a small chance that we miss PoW inputs when by chance blocks are found faster than our PoW request interval.

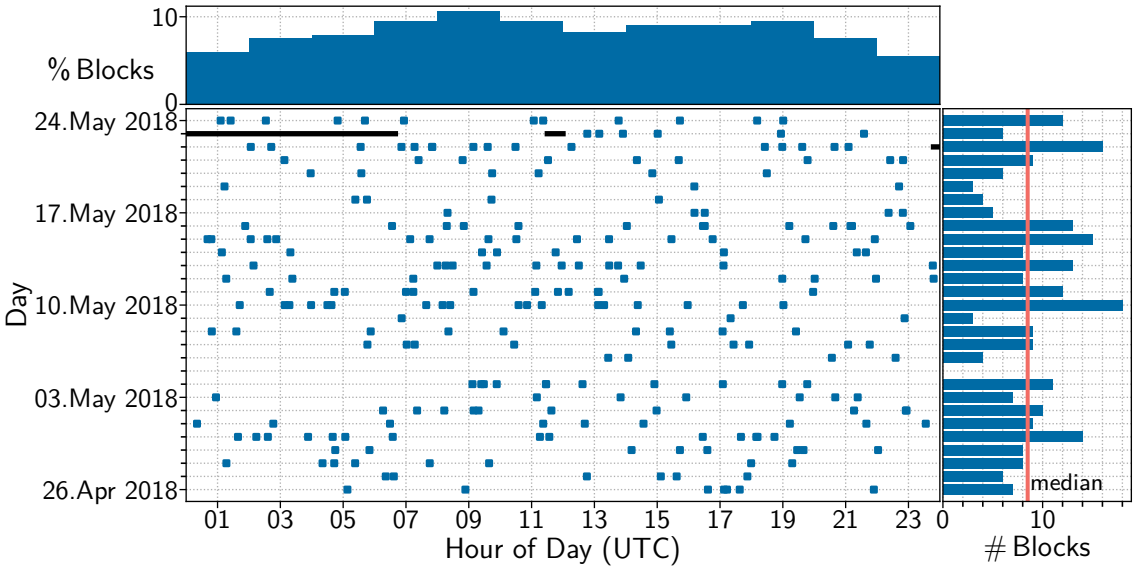


Figure 5.6 Mined blocks over time from the Coinhive network. Black parts mark outages of *our* infrastructure.

hint at the geolocation of the users. However, the figure (upper subplot) shows that blocks are found throughout the whole day, which might be an indicator of Coinhive's global reach. We find multiple days with significantly more blocks than on average, e.g., the 30th of April, 10th, and 22nd of May 2018. The 30th of April precedes Labor Day, a public holiday in over 80 countries, time zone shifts to UTC or holidays could explain increased Internet usage resulting in more mined blocks. Similarly, the latter two were Ascension Day and the day after Pentecost, both public holiday in many (mostly) European countries.

In the median (average), we find 8.5 (9.0) blocks per day, but we noticed a disruption of Coinhive' service on the 6th and 7th of May which resulted in only a few to no announced PoW inputs. We can estimate the combined hash rate of Coinhive by taking the network's difficulty into account. The difficulty denotes the expected number of hashes that a miner must compute (statistically) to find a block. After each block, the network adjusts this number by consensus such that the block rate of two minutes is continued to be met. Throughout our observations, the median difficulty was 55.4G hashes, which translates to a network hash rate of 462M h/s. As Coinhive mines roughly 8.5 blocks per day, they produce 1.18% of all 720 blocks/day which translates to 5.5M h/s. If we assume that a Web client performs between 20 to 100 h/s, then Coinhive requires between 292K and 58K constantly mining users. Comparing our findings with numbers reported by Coinhive [Coi17] from September 2017 is difficult. Coinhive wrote that their hash rate peaked at 13.5M h/s (then 5% of the network's hash rate). However, our results are averages over long periods and derived from the statistical properties of the network, while those published are momentary peak rates, and thus, a direct comparison is not possible.

If we sum up the block rewards of the ultimately mined blocks over the observation period of four weeks, we find that Coinhive earned 1271 XMR. Table 5.6 complements our four-week analysis with three full months in 2018, showing its continuity. Similar

	Median [blocks/day]	Average [blocks/day]	Hashrate [MH/s]	Currency [XMR]
May	9.0	8.8	5.5	1231
June	10.0	9.7	5.5	1293
July	9.0	9.1	5.8	1215

Table 5.6 Coinhive's mining power for three months in 2018.

to other cryptocurrencies, Monero's exchange-rate fluctuates heavily. In August 2018 one XMR was worth 120 USD, having peaked at 500 USD at the beginning of 2018. Thus, assuming 120 USD, Coinhive mines Moneros worth around 150 000 USD per month of which they say, they give 70% to their users. Still, the operational costs seem manageable, making it potentially profitable for Coinhive.

Takeaway. *Coinhive currently contributes ∼1.18% of the mining power of the Monero network. While probably profitable for Coinhive, it remains questionable whether mining is a feasible ad alternative.*

5.1.6 Related Work

Browser-based mining has been subject to substantial media coverage, e.g., reports on Pirate Bay [Gua17a] mining, about hacked websites for mining [War18], miners injected into the Google's DoubleClick ad-platform [Tre18] or drive-by Monero mining on Android [Seg18]. Many blog posts exist that report on Alexa-listed websites to include mining code [Pix17, AdG17]; however, without detailing a methodology. A list published on Github [Sec17] provides an overview of potential mining domains. However, this list also includes entries such as *google.com* which is unlikely to be mining. To the best of our knowledge, parallel to our work, [ELM+18] are the first to investigate browser-based mining in an academic context. While they also find Coinhive to be the most prominent service, their analysis is based on a *string search* over a comprehensive set of HTML bodies, thus not accounting for actual HTML structure which we did in our first analysis (see Section 5.1.2.1) which showed to already produce skewed results, e.g., the categories of websites significantly differs. We thus complement their results by incorporating WebAssembly fingerprinting and further shed light on the inner workings of Coinhive. In further concurrent work, Konoth et al. [KVM+18] estimate revenue for websites that include mining scripts on the Alexa Top 1M. Further, they also analyze the nature of Wasm mining for detection using a very similar methodology to what was the basis for our fingerprinting, such as counting particular instruction or memory access pattern. Based on monitoring DNS, [3Y18] also observes Coinhive as the dominant player. They report that most cryptominers are present on adult websites in the Alexa Top 1K/3K. Similar with regards to link-forwarders, [NMS+14] analyzed ad-based link forwarding services and their revenue model, which relates to that of Coinhive, and thus, we believe that their results also apply here.

5.1.7 Summary and Discussion

This contribution analyzes the prevalence of browser-based mining, a new revenue-generating model to monetize websites and an alternative to ad-based financing that is enabled by ASIC resistant cryptocurrencies such as Monero. By inspecting 137M .com/.net/.org and Alexa Top 1M domains for mining code, we indeed find websites that utilize browser-based mining. We find that this browser-based mining is enabled primarily through the availability of new Web APIs, i.e., Wasm and Websockets. To perform an in-depth inspection of these dynamic Web features, we heavily automate a Web browser to further investigate the use of these new Web APIs. Interestingly, especially Wasm seems to be nearly exclusively used for mining in the Web. While we also find some other instances making use of Wasm, its efficiency is currently clearly abused and likely not in a way that Internet giants that are currently pushing this technology anticipated.

Still, the prevalence of browser mining is *currently* low at $< 0.08\%$ of the probed sites. For its detection, we find that the public NoCoin filter list is insufficient to detect browser mining broadly. We thus present a new technique based on WebAssembly fingerprinting to identify miners, up to 82% of thereby identified mining websites are not detected by blocklists. We identify Coinhive as the largest Web-based mining provider used by 75% of the mining sites. Given its popularity, we further dissect Coinhive's link-forwarding service. We find that ten heavy users contribute over 80% of all short links, mostly targeting streaming and filesharing services. The remaining short links target a diverse set of websites. We continue by dissecting the economics of Coinhive, we devise a new method that allows associating mined blocks with a mining pool, and we find that Coinhive mines 1.18% of all Monero blocks and their visitors have a combined median hash rate of 5.5M h/s. While we find that Coinhive turns around Moneros worth 150 000 USD per month, the current value stability of cryptocurrencies requires further investigations if browser-based mining can be an alternative revenue model to ad-based financing. Further, we expect significant impact of the CPU intensive miner on a mobile device's battery lifetime, and questions how it affects a website's performance or a visitor's energy bill is yet to be quantified, but it could be a huge hurdle to be competitive to ad-based financing on a larger scale.

5.2 Conclusion

In this chapter, we have provided an outlook on how innovation on the application layer can be analyzed in the Web at large. We have found that technologies such as WebAssembly or Websockets are abused to perform cryptomining within a website visitor's browser. While the earnings for individual websites are likely limited, the API provider seems to be able to benefit. Nevertheless, we believe that the technologies such as Wasm or Websockets offer potential beyond cryptomining and that the Internet giants that have been pushing these technologies are not to blame for people abusing them. Still, these giants controlling browsers or infrastructures could set measures into place to restrict the abusive use of the technologies.

Notwithstanding, there are many new Web APIs beyond WebAssembly and Websockets that are being standardized. Most of them have not been analyzed concerning their use beyond their fingerprinting potential. Further, with the introduction of DNS over HTTPS (DoH) [RFC8484] on the application layer and its potential to secure the transport of DNS records while also disrupting the traditional DNS model. Now Web servers deliver DNS records instead of DNS servers, challenging DNS-based monitoring used for malware and SPAM protection while liberating people by circumventing censorship. Further, DNS-based load balancing as profoundly performed by content delivery networks (CDNs) may become even more challenged if DoH messages are intertwined with the regular Web data exchange via HTTPS and Web servers resolve domains on behalf of their visitors while residing in vastly different networks and geolocations. These and other reasons have led to the formation of the *add* working group within the Internet Engineering Task Force (IETF) in mid 2019 [IET19] and offer an exciting research angle to study how Internet evolution may continue.

6

Conclusion

The Internet has undoubtedly lived through a tremendous evolution since the early packet switching. This evolution is best visible in the way how we use the Internet, which, for many parts, has drastically changed since the 1980s, e.g., through the use of video and social media. Already in the early days of the Internet, its inventors observed that there is only one constant in the Internet which is constant change, often driven by new technologies or software innovations. However, today, the Internet is considered ossified around the initial protocol design that dates back to the 1980s, making it very challenging to deploy radical changes. Still, the *visible* evolution driven by new businesses and their corresponding Internet applications has to be supported by a *technological* evolution as the applications today demand increasing bandwidths and reduced latencies that are not supported by the early protocols. In fact, in recent years, we observed the emergence of many new and optimized protocols often driven by the companies behind the demanding applications. Many of these new proposals only become an Internet standard if the company is a giant Internet player such as Google, but even then, many proposals are never deployed or actively used due to the ossification in the network. Thus, the question arises on how the Internet was able to technically evolve subject to the ossification around the initial design given the rising demands.

Methodology and Research Objectives

In this dissertation, we approached this great umbrella question with the help of Internet and network measurements from several angles. Motivated by how Internet giants have transformed the *visibly* used part of the Internet and specifically the Web, we recognize their critical importance today in delivering services, building networks, and their involvement in various standardization bodies. In light of the dependencies of all services on the network layer, transport layer as well as the application layer mechanics, we focus on key technologies that were developed and, at least on paper,

have a critical impact on the performance. To this end, we developed novel Internet measurement methodologies to investigate the large scale use of these technologies in significant parts of the Internet and by Internet giants in particular. Our methods and findings generate insights in Internet evolution and offer a neutral source of information for standardization, research, and education that serves in the design of new protocols or mechanisms.

In the following, we summarize our contributions and findings before we take an outlook into future work and close with some final remarks.

6.1 Contributions and Findings

To approach the continuously unfolding question of how the Internet evolves, we focus on three, in our believe important, questions. These questions are aligned to our observations that Internet giants are transforming the Internet experience as well as the difficulty to actually evolve core Internet technology. The following three sections highlight the initial question and summarize our contributions and findings to answer the questions.

6.1.1 What Is the Impact of Internet Giants on Internet Transport Evolution?

This first question is motivated by the observation that the Transmission Control Protocol (TCP) dominates Internet transport since the 1980s. Moreover, it is this transport that governs the efficiency and performance of Internet transmissions, and thus, plays a vital role in all data transmissions. Given that the wire format of TCP remained unchanged since the 80s, and that changes to it are not deployable, we first focus on the parameterization of the algorithms within TCP. These algorithms, while standardized, are easy to tweak, especially for Internet giants that are in a pursuit for performance. Often the standardized parameterization offers a common ground to enable safe operation of all communication but not necessarily the most efficient operation of individual communication.

Transport Protocol Parameterization

To this end, our first contribution (see Section 3.1) investigates the use of TCP's initial congestion window (IW) in all of Internet Protocol Version 4 (IPv4) and by Internet giants in particular. The IW governs the number of bytes that can be transmitted right at the start of a connection without probing the network and is thus a critical component to bootstrap short Web flows. Our findings show that the Internet Engineering Task Force (IETF)-standardized values prevail in the Internet at large and that the current experimental recommendation of ten segments is prevalent in IPv4. Nevertheless, when focussing on how Internet giants use this parameter, we find values that range between 20 to 32 segments with exceptions of up to 100

segments. These increased values highlight that Internet giants work outside of recommended practices, but we also find that they are well aware of their actions, and some utilize pacing to reduce the change of a negative impact of these increased values. Indeed, within dedicated testbed measurements, we were able to validate the increased performance, especially when using pacing.

QUIC Is the Next Big Thing on the Transport Layer

In our second contribution (see Section 3.2), we focused on Google's QUIC proposal as a radical redesign of the transport layer. QUIC is fueled by decades of transport layer research and promises a clear path towards quickly evolving the transport layer by encrypting even the header data. This resulting lack of observability by Internet service providers (ISPs) challenges the QUIC deployment and standardization. To this end, we were the first to investigate the use of QUIC in all of IPv4, in large parts of the Domain Name System (DNS), and its prevalence in today's traffic mix using traces at a major European Internet exchange point (IXP) and Tier-1 ISP. Our insights highlight that the Google flavor of QUIC is mainly driven by Google and Akamai, which have a massive deployment. Even though the Akamai content delivery network (CDN) fuels significant parts of the Internet, they were, at the time of the investigation, not using it for the mainstream. Looking at the IETF variant of QUIC, we find that the promises to increase the performance have brought further Internet giants on board such as Cloudflare or Facebook. When looking at traffic shares, we found up to 20% of QUIC in various traffic mixes, which are dominated by Google, who push over 50% of their own traffic via QUIC. Thus, we find that QUIC has already transformed the Internet without having left standardization, and thus, is a practical challenge for operators today. We then set out to investigate how much more performance QUIC may give in various emulated network settings. Our findings show that QUIC can indeed outperform TCP, but applying our findings from our first contribution, i.e., how Internet giants parameterize TCP, narrows the gap. In an extensive user study, participants were able to spot a difference between TCP and QUIC in a side-by-side comparison but mostly judge the website loading speed indistinguishable. Only in severely adverse network conditions QUIC seemed to win, but it also showed that often website structure and congestion control (CC) algorithm play a much more significant role than the transport itself.

Congestion Control in the Wild

In our third contribution (see Section 3.3), we further drilled down on CC as the fundamental principle underlying all sensible transports, be it TCP or QUIC. Motivated by our findings and the well-known difficulty of fairness in CC, we set out to investigate how the CC choice of Internet giants affects the resource sharing in the Internet today. To this end, we compare the out-of-box behavior of Linux CC algorithms against Internet giant-originating flows as well as how they share the resources among themselves. Our findings indicate significant levels of unfairness in the Internet today. Some of our findings directly boil down to an explicit choice of CC, e.g., using bottleneck bandwidth and round-trip propagation

time (BBR) instead Cubic as Linux's default. Still, it is too short-sighted to claim that Internet giants abuse this unfairness for several reasons. First, our findings indicate a significant relationship between the network bottleneck and the chosen CC algorithm. For example, choosing BBR while having small network buffers, results in a massive dominance of BBR, but when the buffers get larger, BBR might even be disadvantaged. Second, stealing bandwidth from a competitor might look tempting on paper, but Web content today is made up of a plethora of resources distributed among several servers by several operators. Thus, stealing bandwidth may ultimately have a negative effect, e.g., on the loading speed of a website when competing resources are disadvantaged that are essential for displaying the website. Ultimately, we showed that a simple measure, such as employing a fair active queue management (AQM) algorithm at the receiver side, reintroduces fairness regardless of CC by the Internet giants.

Reflection

Our findings indicate that Internet giants significantly impact and coin Internet transport, be it in its design, deployment, or traffic shares. Their dominance puts them into a powerful position. When they alter portions of their services to use a new protocol or a protocol modification, they significantly impact the Internet as a whole. Furthermore, they have raised the bar for transport standardization, which seems to now require extensive tests of protocol features on significant parts of the Internet before being considered for standardization. Nevertheless, Internet giants open up their innovations and are keen to standardizing them, fueling Internet evolution and protocol deployment. Of course, these actions are not entirely altruistic, and the past has shown that a large deployment also outside the Internet giants is required to overcome ossification problems. Further, while opening up the innovations, their parameterization and configurations, especially in terms of CC, are a business secret and allow further specialization.

6.1.2 How Do Content Owners Flexibilize in Light of Internet Giants and Network Ossification?

This second question is motivated by, *i)* the observed power of Internet giants, and *ii)* the often reported network ossification. More specific, how ossified is the Internet really and how deprecated are the configurations? Further, given that Internet giants seem to work around these limitations, how do content owners do both, i.e., circumvent network innovation limitations while not being at the sole mercy of the Internet giants. We approach this question using two individual contributions that shine a light on both aspects.

Network Ossification and Deprecation

This first contribution (see Section 4.1) towards the second question investigates the degree of ossification and deprecation in the Internet, more specifically in IPv4. If the

Internet were not ossified or suffer from deprecation, there would be a high degree of practical innovations in the Internet. One of the critical challenges of Internet measurements today is to keep the scanning footprint low. Thus, our key idea to study how the Internet ages, in terms of ossification and deprecation, was to put our Internet scans that we used to answer the first question to a second use. To this end, we utilize usually neglected backscatter signals from the Internet's control plane that is fueled by Internet Control Messaging Protocol (ICMP) messages. By capturing this ICMP backscatter to our scans, we were able to paint an expectable but somber picture of the state of the Internet. In summary, we found a significant degree of outdated systems. Ranging from generating long-deprecated source quench (SQ) messages over the generation of redirect messages across the Internet to a high and nuanced degree of unreachability, crowned by a significant extent of routing loops. Our findings show that innovating at the Internet core is a significant challenge as problems documented over two decades ago are still omnipresent today.

Meta-CDNs Liberate Content Provision

In the next contribution (see Section 4.2), we investigated how content owners that seek dissemination can do so in an ossified and deprecated Internet where Internet giants like CDNs control the dissemination infrastructures. To this end, we analyzed the concept of a Meta-CDN at the example of Cedexis, a major Meta-CDN operator. We shed light on Cedexis' customers, technology, and performance. Cedexis DNS deployment appears to be focussing on Europe, North America, and parts of Asia indicated by high latencies in other regions. We find customers to configure mostly low time to live (TTL) values enabling fast reactions, and we indeed observe that most domains balance between few CDNs. By assessing ping latencies to all available CDNs, we observe that most available CDNs offer a good performance to our distributed probing platforms. However, we also find a range of suboptimal latency choices, which can indicate routing metrics other than latency or suboptimal decisions.

Reflection

Our analyses showed that the general public Internet is still highly ossified and obsolete, challenging the deployment of new Internet core functionality, and thus, impedes the general evolution of the Internet. While already CDNs solve many of these impediments on the application layer, e.g., by utilizing DNS indirections, also their globally deployed infrastructure with their deep peering puts them into the situation to evolve more quickly than many others. To this end, it is interesting how Meta-CDNs utilize the same mechanisms to innovate on top of CDNs and challenge their mode of operation. Their model has two angles, one technical and one monetary that naturally follows. That is, it challenges the demand modeling of a CDN and likely also ISPs. The choice in steering traffic to a particular CDN (and hence the revenue) is controlled by the Meta-CDN that can perform these decisions based on costs or performance. Thus, a one-time lousy performance may result in traffic rerouting and losing revenue. Even though we currently believe that the

Meta-CDN principle is not applied universally enough to actually interfere with the CDN operation or cause any harm in traffic engineering (TE) for any involved party. However, if the concept becomes more widely adopted, we might see effects similar to those documented in [BBP+18], where the authors observe how an ISP's TE is overwhelmed by Apple acting like a Meta-CDN during one of their mobile operating system upgrades.

6.1.3 How Are Application Layer Optimizations That Are Pushed Forward by Internet Giants Used at Large?

This last question further looks up the stack where innovation has always been possible without the need for core network support. Especially in the Web that has seen a tremendous visible evolution from static to dynamic and interactive content, many new standards have emerged. Web browser manufacturers such as Google or Mozilla profoundly impact the standardization as usually, support within the browsers is required. Hence, they are strategically positioned to shape the Web landscape in terms of applications enabled by these new standards.

WebAssembly Fuels Browser-based Cryptomining

In our contribution (see Section 5.1) peeking into this broad question, we investigate the use of WebAssembly (Wasm) in the Web. Wasm supersedes the ubiquitously used JavaScript by offering a much more efficient instruction set that closely matches today's computing architectures. Envisioned application areas of Wasm include in-browser gaming, video editing, image recognition, and other computationally heavy tasks. However, driven by media reports about browser-based mining, we found that Wasm is usually at the core of the computationally heavy proof of work (PoW) process governing the mining. By inspecting large parts of the Web, we find that Wasm is rarely used today. Subsequently, we designed a novel methodology to fingerprint the Wasm code finding that about 96% of the collected Wasm code are miners. Further, we found that public blocklists miss up to 82% of them. We then design a methodology to associate the origin of the mined blocks to the largest browser-based cryptocurrency application programming interface (API) vendor. At the time of the study, they had a monthly turn-around of crypto coins worth 150 000 USD, which accounts for over 1% of the whole mining network.

Reflection

Admittingly, we peeked only into the surface of this last question. However, our analyses showed an abuse that was likely unforeseeable by the inventors. While the Internet giants are likely not to blame for the abuse of the technology, also the related field of browser-fingerprinting (challenging a user's privacy through tracking) has shown that currently, Web standardization largely ignores potential adverse effects of their work. We believe that a more thorough risk assessment might be required and that measures, such as demonstrated by our fingerprinting approach, could be

embedded into the standards. For example, by painting a clear path towards how code could be compared, allowing browsers to directly expose such information.

6.2 Future Work

In this dissertation, we came into contact with many technologies and methods that sparked a range of questions and ideas. In the following, we would like to give an overview of some of these that we believe are worth further exploring.

Methodologies to Measure the Evolution in IPv6 and the Web

In our work, we have exclusively focused on IPv4, mostly due to the non-existing Internet Protocol Version 6 (IPv6) infrastructure at our chair. Still, in IPv6, it is structurally infeasible to enumerate all services, and the Internet measurement community is currently working on hitlists, e.g., Gasser et al. [GSG+16]. Furthermore, the increasing use of virtualization in the Internet, that has basically demanded us to modify our IW scanning methodology when investigating CDNs, further diminishes the importance of Internet Protocol (IP) address-based scans. Thus, also the creation of comprehensive and diverse DNS-based lists is ongoing and future work, especially when they should apply to many research areas.

Bootstrapping Congestion Control

Our work on IWs (see Section 3.1) and its observed customization naturally lead to the observation that they could be further tailored to individual users. While there are indeed approaches such as [FKB16] that learn an IW, they also do learn a lower "safe" bound. They are genuinely unaware of the path's capabilities when launching a connection. Depending on where a network bottleneck is located, a client could signal the availability of bandwidth. We have already peeked into this idea in [RWS+19] and added a client-side bandwidth estimate in QUIC, which the server uses to bootstrap CC into congestion avoidance. Still, this approach is prone to false estimates, and it is still unclear what impact such a change would have on more flows and a bottleneck located somewhere else in the network. A promising architecture for investigating the problem in an extended manner is presented by Kühlewind et al. [KBT+17], where on-path nodes can include protected in-band signaling to traversing packets. This way, nodes could signal a safe bandwidth along the path to bootstrap CC as in the Rate Control Protocol (RCP) [Duk07].

Dynamics of Congestion Control in Underutilized Setting

Within all our testbed studies investigating the performance or fairness of CC, we have relied on bulk flows that strive to fully utilize all available bandwidth. Admittingly, many real applications may have a vastly different pattern. For example, the ubiquitous use of video streaming using Hypertext Transfer Protocol (HTTP) DASH

that makes up substantial amounts of Internet traffic leads to a highly bursty CC demand. A burst of data needs to be transmitted following periods of idle. Current CC algorithms lose their network model and should start from scratch (or utilize pacing as is the case for YouTube). However, in the meantime, the network may have changed dramatically, e.g., when assuming mobility. In the advent of QUIC, a redesign of DASH that is tightly coupled to the underlying CC (which becomes accessible in QUIC from the user space) may offer new ways to innovate in this area. In this regard, learning-based CC algorithms, such as Vivace [DMZ+18], might be worth exploring in terms of their cross-layer interactions, and how one can combine their rapid detection of change with the largely predetermined demand of video bitrates.

Congestion Control Identification in the Wild

In our work on CC fairness (see Section 3.3), we easily identified several CC algorithms by how they interacted with flows generated in our testbed. However, some behaved vastly different, and as our work showed, it is detrimental for fairness and performance to test new CC algorithms against those actually used in the Internet. Furthermore, our work on IWs (see Section 3.1) highlights the parameterization potential of CC. Current approaches to CC identification rely on a hand-tailored model of the CC algorithm that fails when only tiny assumptions are rendered invalid, e.g., when the parameterization changes slightly. Further, with QUIC, essential headers for CC identification become unavailable for a passive observer. In this light, we have already worked towards a deep learning-based approach [SRH+19] that solely relies on packet arrival information for CC identification. While it shows already promising results, the dynamics of CC mentioned above also challenge the inference accuracy, and the CC algorithms need to be available to generate training data. Two assumptions that not necessarily hold in the general Internet and warrant further research. Especially the application of machine learning to networking tasks, may not only in this specific setting offer further means to handle the vast diversity of the Internet.

User Privacy in QUIC

QUIC promises increased privacy through pervasive encryption. Our work on fingerprinting services with the help of connection parameters (see Section 3.2.4.2), however, highlights that privacy-exposing mechanisms may still be within the protocol. On a larger scale, we believe it is of great interest to methodologically investigate how an active “attacker” could expose a user’s privacy. A starting angle for this could be QUIC’s acknowledgment (ACK) delay mechanism, encoding the time it took from receiving a packet to generating the corresponding ACK. This timing information could be unique to individual users and devices or within specific load scenarios such that blinding of these values may be required to preserve privacy. Ultimately, a methodological approach in finding such hidden entropy exposing user privacy is an ambitious research objective.

Understanding the Dark Net

In this dissertation, we have utilized large-scale Internet measurements, e.g., in our investigations of Wasm usage (see Section 5.1), we scanned significant parts of the Web. A largely neglected portion of the Internet in the area of Internet measurements is the Dark Net building on top of Tor. While there are a couple of studies [BPW13, BPT+14, AFA+19] that utilize long-fixed vulnerabilities in Tor or public lists of services, a fundamental understanding of the nature of this network is missing in academia. This has a plethora of angles, for once, Tor itself still suffers from high latencies and low performance. To this end, Dhungel et al. [DSR+10] have shown significant levels of delay within Tor; however, their study nearly dates back ten years, and much has changed since then within Tor. Such an understanding ties to the use of CC and its dynamics between the overlay nodes in Tor and could offer exciting research objectives when combining it with the rich capabilities of QUIC, e.g., mapping circuits to QUIC streams, reducing the effects of loss across circuits. Furthermore, websites hosted as a hidden service are known to be rather simple due to the low performance of Tor itself. Still, an in-depth analysis of their structural differences is missing. Similarly, the Dark Net is often synonymous with a Dark Web, but in fact, the Dark Net is not limited to websites demanding an in-depth analysis of other services in Tor. From a research perspective, a tool similar to ZMap would enable diverse research in the Dark Net that could be used to investigate its evolution but has huge hurdles as Tor is designed to avoid such an enumeration.

Effects of Upcoming Internet Standards on Internet Evolution

This dissertation offers a snapshot of Internet evolution spanning a couple of years. Undoubtedly, the Internet will continue to evolve, new players will emerge, and others will fade away. In light of this constant change, we believe that our work investigating the use of new Web APIs may offer many additional angles for future work as this area is rapidly evolving. Similarly, fundamental changes such as our analysis of QUIC (see Section 3.2) naturally offer significant potential for future work. Along these lines is DNS over HTTP Secure (DoH) that currently sees a deployment that warrants investigations, e.g., are techniques such as website fingerprinting, as applied in Tor, applicable to DoH at the ISP to derive the DNS query?

6.3 Concluding Remarks

In this dissertation, we strived to enlighten Internet evolution with the help of large-scale Internet measurements. Our goal was to gain a better understanding of how the Internet practically works, and how the working principles have deviated from textbook practice as well as how the discovered principles affect Internet operation. While we are confident that the Internet continues to evolve and that this dissertation can only serve as a snapshot in Internet history, we believe that the individual contributions nevertheless advanced the general understanding of the Internet. To this end, we have presented our work at various international conferences

and standardization meetings, e.g., at the IETF-101 or IETF-105, which allowed us to get feedback from the protocol engineers and fuel the discussion around new protocols and mechanisms, e.g., in QUIC as well as regarding HTTP/2 Server Push.

Still, this feedback and the perspectives that we have gathered have shown us that there is always more to pursue and opened our understanding of the Internet beyond the contributions of this thesis. Thus, we are eager to experience how the Internet continues to evolve and how the future research angles that we outlined will affect this evolution. In this regard, we close this dissertation with a well-aged quote from the Internet's founding fathers.

> *New modes of access and new forms of service will spawn new applications, which in turn will drive further evolution of the net itself.*
>
> *The most pressing question for the future of the Internet is not how the technology will change, but how the process of change and evolution itself will be managed.*
>
> — *Barry M. Leiner et al. [LCC+09]*

Abbreviations and Acronyms

ACK	Acknowledgment
AEAD	Authenticated Encryption with Associated Data
AFCT	Average FCT
AIMD	Additive-Increase Multiplicative-Decrease
AMSS	Aeronautical Mobile-Satellite Service
ANOVA	Analysis Of Variance
API	Application Programming Interface
AQM	Active Queue Management
ARPANET	Advanced Research Projects Agency Network
AS	Autonomous System
ASIC	Application-Specific Integrated Circuit
ASN	AS Number
BBR	Bottleneck Bandwidth and Round-Trip Propagation Time
BDP	Bandwidth Delay Product
BGP	Boarder Gateway Protocol
CC	Congestion Control
CCDF	Complementary CDF
CDF	Cumulative Distribution Function
CDI	Content Distribution Infrastructure
CDN	Content Delivery Network
CFCW	Connection Flow Control Receive Window
CNAME	Canonical Name
CoDel	Controlled Delay
CP	Content Provider
CPU	Central Processing Unit
cwnd	Congestion Window
DA2GC	Direct Air-To-Ground Communications
DNS	Domain Name System
DoH	DNS over HTTPS
DOM	Document Object Model
DRR	Deficit Round-Robin
DSL	Digital Subscriber Line
ECMP	Equal-Cost Multi-Path
ECN	Explicit Congestion Notification
FCT	Flow Completion Time
FPGA	Field-Programmable Gate Array
FQ	Flow Queuing
FVC	First Visual Change
GPU	Graphics Processing Unit
gQUIC	Google QUIC
HoL	Head-of-Line
HTML	Hypertext Markup Language
HTTPS	HTTP Secure
HTTP	Hypertext Transfer Protocol
ICANN	Internet Corporation for Assigned Names and Numbers
ICMP	Internet Control Messaging Protocol
ICSL	Idle Connection State Lifetime

IDS Intrusion Detection System
IETF Internet Engineering Task Force
IGP Interior Gateway Protocol
IP Internet Protocol
IPv4 IP Version 4
IPv6 IP Version 6
iQUIC IETF QUIC
ISP Internet Service Provider
IW Initial Congestion Window
IXP Internet Exchange Point
KPI Key Performance Indicator
LTE Long Term Evolution
LVC Last Visual Change
MIDS Maximum Incoming Dynamic Streams
MIT Massachusetts Institute Of Technology
MPLS Multiprotocol Label Switching
MSS Maximum Segment Size
MTU Maximum Transmission Unit
NAT Network Address Translation
NIC Network Interface Card
NS Name Server
PLT Page Load Time
PMTU Path Maximum Transmission Unit
PoW Proof of Work
QoE Quality of Experience
RACK Recent Acknowledgment
RCP Rate Control Protocol
rDNS Reverse DNS
RED Random Early Detection
RFC Request for Comments
RIR Regional Internet Registry
RTO Retransmission Timeout
RTT Round-Trip Time
rwnd Receive Window
SACK Selective Acknowledgment
SCFG Server Config
SFCW Stream Flow Control Receive Window
SI Speed Index
SLA Service Level Agreement
SNI Server Name Indication
SQ Source Quench
sRTT Smoothed RTT
ssthresh Slow Start Threshold
STK Source-Address Token
SVID Server ID
TC Traffic Control
TCB Transmission Control Block
TCP Transmission Control Protocol
TE Traffic Engineering
TLD Top-Level Domain
TLS Transport Layer Security
TTL Time to Live
UDP User Datagram Protocol
URI Uniform Resource Identifier
URL Uniform Resource Locator
VC85 Visual Completeness 85%
VPN Virtual Private Network
W3C World Wide Web Consortium
Wasm WebAssembly

Bibliography

[3Y18] 360Netlabs and Xu Yang. Who is Stealing My Power: Web Mining Domains Measurement via DNSMon, 2018. URL: https://web.archive.org/web/20180515/http://blog.netlab.360.com/who-is-stealing-my-power-web-mining-domains-measurement-via-dnsmon-en/ (archived on 2018-05-15).

[AdG17] AdGuard. Cryptocurrency mining affects over 500 million people. And they have no idea it is happening., October 12, 2017. URL: https://web.archive.org/web/20180515/https://adguard.com/en/blog/crypto-mining-fever/ (archived on 2018-05-15).

[AGH+12] Vijay Kumar Adhikari, Yang Guo, Fang Hao, Volker Hilt, and Zhi-Li Zhang. A Tale of Three CDNs: An Active Measurement Study of Hulu and Its CDNs. In *Proceedings of the INFOCOM Global Internet Symposium (GI '12)*, pages 7–12. IEEE, 2012. DOI: 10.1109/INFCOMW.2012.6193524.

[ACF+12] Bernhard Ager, Nikolaos Chatzis, Anja Feldmann, Nadi Sarrar, Steve Uhlig, and Walter Willinger. Anatomy of a Large European IXP. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '12)*, pages 163–174. ACM, 2012. DOI: 10.1145/2342356.2342393.

[ASA00] A. Aggarwal, S. Savage, and T. Anderson. Understanding the Performance of TCP Pacing. In *Proceedings of the International Conference on Computer Communications (INFOCOM '00)*, pages 1157–1165. IEEE, 2000. DOI: 10.1109/INFCOM.2000.832483.

[Aka16] Akamai. Q4 2016 State of the Internet - Connectivity Report, March 1, 2016. URL: https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/q4-2016-state-of-the-internet-connectivity-report.pdf (visited on 2019-06-30).

[AC18] Akamai and Dave Comery. FAQ: QUIC Native Platform Support for Media Delivery Products, April 10, 2018. URL: https://community.akamai.com/customers/s/article/FAQ-QUIC-Native-Platform-Support-for-Media-Delivery-Products (visited on 2019-08-01).

[Aka15] Akamai Community Forum. Can we change initial CWIN for web experience products like DSA, Ion?, July 28, 2015. URL: https://web.archive.org/web/20180510/https://community.akamai.com/thread/2694 (archived on 2018-05-10).

[All15] Mark Allman. Removing TCP's Initial Congestion Window. Internet-Draft draft-allman-tcpm-no-initwin-00.txt, Internet Engineering Task Force, November 24, 2015, pages 1–3. URL: https://datatracker.ietf.org/doc/html/draft-allman-tcpm-no-initwin-00. Work in Progress.

[Ama19] Amazon.com, Inc. Amazon Web Service IP Address Ranges, August 20, 2019. URL: https://web.archive.org/web/20190821/https://ip-ranges.amazonaws.com/ip-ranges.json (archived on 2019-08-21).

[AG18] AOL and Google. WebPagetest, 2018. URL: https://web.archive.org/web/20190821/https://www.webpagetest.org/ (archived on 2019-08-21).

[AKM04] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. Sizing Router Buffers. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '04)*, pages 281–292. ACM, 2004. DOI: 10.1145/1030194.1015499.

[ACO+06] Brice Augustin, Xavier Cuvellier, Benjamin Orgogozo, Fabien Viger, Timur Friedman, Matthieu Latapy, Clémence Magnien, and Renata Teixeira. Avoiding Traceroute Anomalies with Paris Traceroute. In *Proceedings of the Internet Measurement Conference (IMC '06)*, pages 153–158. ACM, 2006. DOI: 10.1145/1177080.1177100.

[BRJ+18] Shehar Bano, Philipp Richter, Mobin Javed, Srikanth Sundaresan, Zakir Durumeric, Steven J. Murdoch, Richard Mortier, and Vern Paxson. Scanning the Internet for Liveness. *SIGCOMM Computer Communication Review (CCR Apr. '18)*, 48(2):2–9, ACM, April 2018. DOI: 10.1145/3213232.3213234.

[BCL09] Steven Bauer, David D. Clark, and William Lehr. The Evolution of Internet Congestion. In *Proceedings of the Research Conference on Communication, Information and Internet Policy (TPRC '09)*, pages 1–34. SSRN, 2009. URL: http://ssrn.com/abstract=1999830.

[BPT+14] Alex Biryukov, Ivan Pustogarov, Fabrice Thill, and Ralf-Philipp Weinmann. Content and Popularity Analysis of Tor Hidden Services. In *Proceedings of the International Conference on Distributed Computing Systems Workshops (ICDCSW '14)*, pages 188–193. IEEE, 2014. DOI: 10.1109/ICDCSW.2014.20.

[BPW13] Alex Biryukov, Ivan Pustogarov, and Ralf-Philipp Weinmann. Trawling for Tor Hidden Services: Detection, Measurement, Deanonymization. In *Proceedings of the Symposium on Security and Privacy (S&P '13)*, pages 80–94. IEEE Computer Society, 2013. DOI: 10.1109/SP.2013.15.

[BG16] Prasenjeet Biswal and Omprakash Gnawali. Does QUIC Make the Web Faster? In *Proceedings of the Global Communications Conference (GLOBECOM '16)*, pages 1–6. IEEE, 2016. DOI: 10.1109/GLOCOM.2016.7841749.

[BBP+18] Jeremias Blendin, Fabrice Bendfeldt, Ingmar Poese, Boris Koldehofe, and Oliver Hohlfeld. Dissecting Apple's Meta-CDN During an iOS Update. In *Proceedings of the Internet Measurement Conference (IMC '18)*, pages 408–414. ACM, 2018. DOI: 10.1145/3278532.3278567.

[BDM+17] Enrico Bocchi, Luca De Cicco, Marco Mellia, and Dario Rossi. The Web, the Users, and the MOS: Influence of HTTP/2 on User Experience. In *Proceedings of the Conference on Passive and Active Measurement (PAM '17)*, pages 47–59. Springer, Cham, 2017. DOI: 10.1007/978-3-319-54328-4_4.

[Bor15] Christian Borman. TCP Evolution: Measuring the deployment of the TCP Evolution. Master's Thesis, Rheinisch-Westfälische Technische Hochschule Aachen, December 2015.

[Bra17] Lawrence Brakmo. TCP-BPF: Programmatically tuning TCP behavior through BPF. In *Proceedings of the Technical Conference on Linux Networking (NetDev 2.2)*, pages 1–5, 2017. URL: https://netdevconf.org/2.2/papers/brakmo-tcpbpf-talk.pdf.

[Bri07] Bob Briscoe. Flow Rate Fairness: Dismantling a Religion. *SIGCOMM Computer Communication Review (CCR Mar. '07)*, 37(2):63–74, ACM, March 2007. DOI: 10.1145/1232919.1232926.

[BAM11] Jake Brutlag, Zoe Abrams, and Pat Meenan. Above the Fold Time: Measuring Web Page Performance Visually, March 15, 2011. URL: https://web.archive.org/web/20190910/https://conferences.oreilly.com/velocity/velocity-mar2011/public/schedule/detail/18692 (archived on 2019-09-10).

[CFK+15] Matt Calder, Ashley Flavel, Ethan Katz-Bassett, Ratul Mahajan, and Jitendra Padhye. Analyzing the Performance of an Anycast CDN. In *Proceedings of the Internet Measurement Conference (IMC '15)*, pages 531–537, Tokyo, Japan. ACM, 2015. DOI: 10.1145/2815675.2815717.

[Car17] Neal Cardwell. BBR evaluation with netem, April 23, 2017. URL: https://web.archive.org/web/20190604/https://groups.google.com/forum/message/raw?msg=bbr-dev/8LYkNt17V_8/xyZZCwcnAwAJ (archived on 2019-06-04).

[CCG+16a] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. BBR Congestion Control, November 15, 2016. URL: https://www.ietf.org/proceedings/97/slides/slides-97-iccrg-bbr-congestion-control-02.pdf (visited on 2019-08-06).

[CCG+16b] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. BBR: Congestion-Based Congestion Control. *ACM Queue (QUEUE Sept.-Oct. '16)*, 14(5):20–53, ACM, October 2016. DOI: 10.1145/3012426.3022184.

[CCY+17] Neal Cardwell, Yuchung Cheng, Soheil Hassas Yeganeh, and Van Jacobson. BBR Congestion Control. Internet-Draft draft-cardwell-iccrg-bbr-congestion-control-00, Internet Engineering Task Force, July 3, 2017, pages 1–34. URL: https://datatracker.ietf.org/doc/html/draft-cardwell-iccrg-bbr-congestion-control-00. Work in Progress.

[CSA+18] Esteban Carisimo, Carlos Selmo, J. Ignacio Alvarez-Hamelin, and Amogh Dhamdhere. Studying the Evolution of Content Providers in the Internet Core. In *Proceedings of the IFIP Network Traffic Measurement and Analysis Conference (TMA '18)*, pages 1–8. IEEE, 2018. DOI: 10.23919/TMA.2018.8506513.

[CDM15] Gaetano Carlucci, Luca De Cicco, and Saverio Mascolo. HTTP over UDP: An Experimental Investigation of QUIC. In *Proceedings of the Symposium on Applied Computing (SAC '15)*, pages 609–614. ACM, 2015. DOI: 10.1145/2695664.2695706.

[CDN17] CDNPlanet. Initcwnd settings of major CDN providers, February 13, 2017. URL: https://web.archive.org/web/20190821/https://www.cdnplanet.com/blog/initcwnd-settings-major-cdn-providers/ (archived on 2019-08-21).

[CJ17] Cedexis and Simon Jones. Cedexis Blog: DDoS Attack, May 11, 2017. URL: https://web.archive.org/web/20180929/www.cedexis.com/blog/ddos-attack-details/ (archived on 2018-09-29).

[CST15] Fangfei Chen, Ramesh K. Sitaraman, and Marcelo Torres. End-User Mapping: Next Generation Request Routing for Content Delivery. In *Proceedings of the ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*, pages 167–181, London, United Kingdom. ACM, 2015. DOI: 10.1145/2785956.2787500.

[CCD+19] Yuchung Cheng, Neal Cardwell, Nandita Dukkipati, and Priyaranjan Jha. RACK: a time-based fast loss detection algorithm for TCP. Internet-Draft draft-ietf-tcpm-rack-05, Internet Engineering Task Force, May 26, 2019, pages 1–29. URL: https://datatracker.ietf.org/doc/html/draft-ietf-tcpm-rack-05. Work in Progress.

[Chr18a] ChromeDevTools. DevTools Protocol API docs – its domains, methods, and events, May 17, 2018. URL: https://web.archive.org/web/20180517/https://github.com/ChromeDevTools/debugger-protocol-viewer (archived on 2018-05-17).

[Chr18b] The Chromium Authors. Add QUIC v44 which will use IETF header format (4db9ea161624970888d73ed15bf4a38e60ae813c), June 8, 2018. URL: http://web.archive.org/web/20190801/https://chromium.googlesource.com/chromium/src/+/4db9ea161624970888d73ed15bf4a38e60ae813c (archived on 2019-08-01).

[Chr18c] The Chromium Authors. QUIC v40 Bug: quic_versions.h (929883afba6805987954c3628a66ae5de9ea9a32) — line 92, September 7, 2018. URL: https://web.archive.org/web/20190801/https://chromium.googlesource.com/chromium/src/+/929883afba6805987954c3628a66ae5de9ea9a32/net/third_party/quic/core/quic_versions.h (archived on 2019-08-01).

[Cis08] Cisco Systems, Inc. IP Routing Frequently Asked Questions, April 30, 2008. URL: https://web.archive.org/web/20190813/https://www.cisco.com/c/en/us/support/docs/ip/border-gateway-protocol-bgp/28745-44.html (archived on 2019-08-13).

[Cis14] Cisco Systems, Inc. Cisco IOS XR MPLS: mpls ip-ttl-propagate, July 28, 2014. URL: https://web.archive.org/web/20190813/https://www.cisco.com/c/en/us/td/docs/routers/xr12000/software/xr12k_r4-1/mpls/command/reference/b_mpls_cr41xr12k/b_mpls_cr41xr12k_chapter_010.html (archived on 2019-08-13).

[Cis16] Cisco Umbrella. Cisco Umbrella List of Top 1M Domains, 2016. URL: https://web.archive.org/web/20190814/http://s3-us-west-1.amazonaws.com/umbrella-static/index.html (visited on 2019-08-14).

[Cit18] Citrix Systems, Inc. Cedexis, 2018. URL: https://web.archive.org/web/20181026/https://www.cedexis.com/ (archived on 2018-10-26).

[Cle19] Lucas Clemente. quic-go — A QUIC implementation in pure go, July 31, 2019. URL: https://web.archive.org/web/20190731/https://github.com/lucas-clemente/quic-go (archived on 2019-07-31).

[Coi17] Coinhive. First Week Status Report, 2017. URL: https://web.archive.org/web/20180515/https://coinhive.com/blog/status-report (archived on 2018-05-15).

[Coi18] Coinhive. Coinhive – Monero JavaScript Mining, 2018. URL: https://web.archive.org/web/20180515/https://coinhive.com/ (archived on 2018-05-15).

[Con19] Conviva Inc. Video AI Platform – Measurement and analytics for the next generation of TV | Conviva, 2019. URL: https://web.archive.org/web/20190814/https://www.conviva.com/ (archived on 2019-08-14).

[CMT+17] Sarah Cook, Bertrand Mathieu, Patrick Truong, and Isabelle Hamchaoui. QUIC: Better For What and For Whom? In *Proceedings of the International Conference on Communications (ICC '17)*, pages 1–6. IEEE, 2017. DOI: 10.1109/ICC.2017.7997281.

[Cry18] Crypto-Loot. Crypto-Loot - A Web Browser Miner | Traffic Miner | CoinHive Alternative, 2018. URL: https://web.archive.org/web/20180515/https://crypto-loot.com/ (archived on 2018-05-15).

[CFL18] Ana Custura, Gorry Fairhurst, and Iain Learmonth. Exploring usable Path MTU in the Internet. In *Proceedings of the IFIP Network Traffic Measurement and Analysis Conference (TMA '18)*, pages 1–8. IEEE, 2018. DOI: 10.23919/TMA.2018.8506538.

[CAZ+14] Jakub Czyz, Mark Allman, Jing Zhang, Scott Iekel-Johnson, Eric Osterweil, and Michael Bailey. Measuring IPv6 Adoption. In *Proceedings of the Conference on SIGCOMM (SIGCOMM '14)*, pages 87–98. ACM, 2014. DOI: 10.1145/2619239.2626295.

[DBK+18] Piet De Vaere, Tobias Bühler, Mirja Kühlewind, and Brian Trammell. Three Bits Suffice: Explicit Support for Passive Measurement of Internet Latency in QUIC and TCP. In *Proceedings of the Internet Measurement Conference (IMC '18)*, pages 22–28. ACM, 2018. DOI: 10.1145/3278532.3278535.

[Dee98] Steve Deering. ICNP '98 Keynote: Watching the Waist of the Protocol Hourglass, October 14, 1998. URL: https://www.ieee-icnp.org/1998/Keynote.ppt (visited on 2019-06-05).

[DeV17] Robert DeVoe. Tombs.io Launches Collaborative Online Game Powered by Monero Mining, December 19, 2017. URL: https://web.archive.org/web/20180516/https://btcmanager.com/tombs-io-launches-collaborative-online-game-powered-monero-mining/ (archived on 2018-05-16).

[DCG+18] Amogh Dhamdhere, David D. Clark, Alexander Gamero-Garrido, Matthew Luckie, Ricky K. P. Mok, Gautam Akiwate, Kabir Gogia, Vaibhav Bajpai, Alex C. Snoeren, and kc claffy. Inferring Persistent Interdomain Congestion. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '18)*, pages 1–15. ACM, 2018. DOI: 10.1145/3230543.3230549.

[DLH+12] Amogh Dhamdhere, Matthew Luckie, Bradley Huffaker, kc claffy, Ahmed Elmokashfi, and Emile Aben. Measuring the Deployment of IPv6: Topology, Routing and Performance. In *Proceedings of the Internet Measurement Conference (IMC '12)*, pages 537–550. ACM, 2012. DOI: 10.1145/2398776.2398832.

[DSR+10] Prithula Dhungel, Moritz Steiner, Ivica Rimac, Volker Hilt, and Keith W. Ross. Waiting for Anonymity: Understanding Delays in the Tor Overlay. In *Proceedings of the International Conference on Peer-to-Peer Computing (P2P '10)*, pages 1–4. IEEE, 2010. DOI: 10.1109/P2P.2010.5569995.

[DMP+02] John Dilley, Bruce Maggs, Jay Parikh, Harald Prokop, Ramesh Sitaraman, and Bill Weihl. Globally Distributed Content Delivery. *IEEE Internet Computing (IC Sep.-Oct. '02)*, 6(5):50–58, IEEE, September 2002. DOI: 10.1109/MIC.2002.1036038.

[DSA+11] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. Understanding the Impact of Video Quality on User Engagement. In *Proceedings of the SIGCOMM Conference (SIGCOMM '11)*, pages 362–373, Toronto, Ontario, Canada. ACM, 2011. DOI: 10.1145/2018436.2018478.

[DMZ+18] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. PCC Vivace: Online-Learning Congestion Control. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI '18)*, pages 343–356. USENIX, 2018. URL: https://www.usenix.org/node/211246.

[DLM+12] Benoit Donnet, Matthew Luckie, Pascal Mérindol, and Jean-Jacques Pansiot. Revealing MPLS Tunnels Obscured from Traceroute. *SIGCOMM Computer Communication Review (CCR Mar. '12)*, 42(2):87–93, ACM, March 2012. DOI: 10.1145/2185376.2185388.

[Duk07] Nandita Dukkipati. Rate Control Protocol (RCP): Congestion control to make flows complete quickly. Ph.D. Thesis, Stanford University, 2007. URL: http://yuba.stanford.edu/~nanditad/thesis-NanditaD.pdf (visited on 2019-10-29).

[DDM10] Nandita Dukkipati, Eric Dumazet, and David S. Miller. TCP: increase default initial receive window., December 20, 2010. DOI: 10.5281/zenodo.1246469.

[DRC+10] Nandita Dukkipati, Tiziana Refice, Yuchung Cheng, Jerry Chu, Tom Herbert, Amit Agarwal, Arvind Jain, and Natalia Sutin. An Argument for Increasing TCP's Initial Congestion Window. *SIGCOMM Computer Communication Review (CCR July '10)*, 40(3):26–33, ACM, July 2010. DOI: 10.1145/1823844.1823848.

[DLK+14] Zakir Durumeric, Frank Li, James Kasten, Johanna Amann, Jethro Beekman, Mathias Payer, Nicolas Weaver, David Adrian, Vern Paxson, Michael Bailey, and J. Alex Halderman. The Matter of Heartbleed. In *Proceedings of the Internet Measurement Conference (IMC '14)*, pages 475–488. ACM, 2014. DOI: 10.1145/2663716.2663755.

[DWH13] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. ZMap: Fast Internet-wide Scanning and Its Security Applications. In *Proceedings of USENIX Conference on Security (USENIX Sec '13)*, pages 1–16. USENIX Association, 2013. URL: https://www.usenix.org/conference/usenixsecurity13/technical-sessions/paper/durumeric.

[Edd19] Wesley Eddy. Transmission Control Protocol Specification. Internet-Draft draft-ietf-tcpm-rfc793bis-13, Internet Engineering Task Force, June 3, 2019, pages 1–104. URL: https://datatracker.ietf.org/doc/html/draft-ietf-tcpm-rfc793bis-13. Work in Progress.

[EKT+17] Korian Edeline, Mirja Kühlewind, Brian Trammell, and Benoit Donnet. Copycat: Testing Differential Treatment of New Transport Protocols in the Wild. In *Proceedings of the Applied Networking Research Workshop (ANRW '17)*, pages 13–19. ACM, 2017. DOI: 10.1145/3106328.3106330.

[EGG+06] Mihaela Enachescu, Yashar Ganjali, Ashish Goel, Nick McKeown, and Tim Roughgarden. Routers with Very Small Buffers. In *Proceedings of the International Conference on Computer Communications (INFOCOM '06)*, pages 1–11. IEEE, 2006. DOI: 10.1109/INFOCOM.2006.240.

[EGR+11] Jeffrey Erman, Alexandre Gerber, K. K. Ramadrishnan, Subhabrata Sen, and Oliver Spatscheck. Over the Top Video: The Gorilla in Cellular Networks. In *Proceedings of the Internet Measurement Conference (IMC '11)*, pages 127–136. ACM, 2011. DOI: 10.1145/2068816.2068829.

[ELM+18] Shayan Eskandari, Andreas Leoutsarakos, Troy Mursch, and Jeremy Clark. A first look at browser-based Cryptojacking. In *Proceedings of the European Symposium on Security and Privacy Workshops (EuroS&PW '18)*, pages 58–66. IEEE, 2018. DOI: 10.1109/EuroSPW.2018.00014.

[FJT+19] Gorry Fairhurst, Tom Jones, Michael Tüxen, Irene Ruengeler, and Timo Voelker. Packetization Layer Path MTU Discovery for Datagram Transports. Internet-Draft draft-ietf-tsvwg-datagram-plpmtud-08, Internet Engineering Task Force, June 5, 2019, pages 1–41. URL: https://tools.ietf.org/html/draft-ietf-tsvwg-datagram-plpmtud-08. Work in Progress.

[Fin89] Gregory G. Finn. A Connectionless Congestion Control Algorithm. *SIGCOMM Computer Communication Review (CCR Oct. '89)*, 19(5):12–31, ACM, October 1989. DOI: 10.1145/74681.74683.

[FG14] Marc Fischlin and Felix Günther. Multi-Stage Key Exchange and the Case of Google's QUIC Protocol. In *Proceedings of the Conference on Computer and Communications Security (CCS '14)*, pages 1193–1204. ACM, 2014. DOI: 10.1145/2660267.2660308.

[FG17] Marc Fischlin and Felix Günther. Replay Attacks on Zero Round-Trip Time: The Case of the TLS 1.3 Handshake Candidates. In *Proceedings of the European Symposium on Security and Privacy (EuroS&P '18)*, pages 60–75. IEEE, 2017. DOI: 10.1109/EuroSP.2017.18.

[FMM+15] Ashley Flavel, Pradeepkumar Mani, David Maltz, Nick Holt, Jie Liu, Yingying Chen, and Oleg Surmachev. FastRoute: A Scalable Load-Aware Anycast Routing Architecture for Modern CDNs. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI '15)*, pages 381–394. USENIX Association, 2015. URL: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/flavel.

[FKB16] Marcel Flores, Amir R. Khakpour, and Harkeerat Bedi. Riptide: Jump-Starting Back-Office Connections in Cloud Systems. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS '16)*, pages 78–87. IEEE, 2016. DOI: 10.1109/ICDCS.2016.47.

[Flo94] Sally Floyd. TCP and Explicit Congestion Notification. *SIGCOMM Computer Communication Review (CCR Oct. '94)*, 24(5):8–23, ACM, October 1994. DOI: 10.1145/205511.205512.

[FJ93] Sally Floyd and Van Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking (ToN Aug. '93)*, 1(4):397–413, IEEE Press, August 1993. DOI: 10.1109/90.251892.

[FK03] Sally Floyd and Eddie Kohler. Internet Research Needs Better Models. *SIGCOMM Computer Communication Review (CCR Jan. '03)*, 33(1):29–34, ACM, January 2003. DOI: 10.1145/774763.774767.

[FB07] Pierre Francois and Olivier Bonaventure. Avoiding Transient Loops During the Convergence of Link-State Routing Protocols. *IEEE/ACM Transactions on Networking (ToN Dec. '07)*, 15(6):1280–1292, IEEE Press, December 2007. DOI: 10.1109/TNET.2007.902686.

[FPL+13] Benjamin Frank, Ingmar Poese, Yin Lin, Georgios Smaragdakis, Anja Feldmann, Bruce Maggs, Jannis Rake, Steve Uhlig, and Rick Weber. Pushing CDN-ISP collaboration to the limit. *SIGCOMM Computer Communication Review (CCR July '13)*, 43(3):34–44, ACM, July 2013. DOI: 10.1145/2500098.2500103.

[GM06] Yashar Ganjali and Nick McKeown. Update on Buffer Sizing in Internet Routers. *SIGCOMM Computer Communication Review (CCR Oct. 06)*, 36(5):67–70, ACM, October 2006. DOI: 10.1145/1163593.1163605.

[GDA17] Qingzhu Gao, Prasenjit Dey, and Parvez Ahammad. Perceived Performance of Top Retail Webpages In the Wild: Insights from Large-scale Crowdsourcing of Above-the-Fold QoE. In *Proceedings of the Workshop on QoE-based Analysis and Management of Data Communication Networks (Internet QoE '17)*, pages 13–18. ACM, 2017. DOI: 10.1145/3098603.3098606.

[GSG+16] Oliver Gasser, Quirin Scheitle, Sebastian Gebhard, and Georg Carle. Scanning the IPv6 Internet: Towards a Comprehensive Hitlist. In *Proceedings of International Workshop on Traffic Monitoring and Analysis (TMA '16)*, pages 1–9. IFIP, 2016. URL: https://tma.ifip.org/2016/papers/tma2016-final51.pdf (visited on 2019-10-15).

[GD11] Alexandre Gerber and Robert Doverspike. Traffic Types and Growth in Backbone Networks. In *Proceedings of the Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference (OFC/NFOEC '11)*, pages 1–3. IEEE, 2011. URL: https://ieeexplore.ieee.org/document/5875624.

[GN12] Jim Gettys and Kathleen Nichols. Bufferbloat: Dark Buffers in the Internet. *Communications of the ACM (CACM Jan. '12)*, 55(1):57–65, ACM, January 2012. DOI: 10.1145/2063176.2063196.

[GAL+08] Phillipa Gill, Martin Arlitt, Zongpeng Li, and Anirban Mahanti. The Flattening Internet Topology: Natural Evolution, Unsightly Barnacles or Contrived Collapse? In *Proceedings of the Conference on Passive and Active Measurement (PAM '08)*, pages 1–10. Springer, Berlin, Heidelberg, 2008. DOI: 10.1007/978-3-540-79232-1_1.

[Gil02] Stephen Gill. ICMP redirects are ba'ad, mkay? Version 1.1, Team Cymru, Inc., July 23, 2002. URL: https://www.cymru.com/gillsr/documents/icmp-redirects-are-bad.pdf (visited on 2019-08-13).

[Goo18] Google. WebPagetest CDN domain list, cdn.h, January 30, 2018. URL: https://web.archive.org/web/20190821/https://github.com/WPO-Foundation/webpagetest/blob/18.10/agent/wpthook/cdn.h (archived on 2019-08-21).

[Goo19a] Google. Google IPv6 Adoption Statistics, August 11, 2019. URL: https://web.archive.org/web/20190813/https://www.google.com/intl/en/ipv6/statistics.html (archived on 2019-08-13).

[Goo19b] Google. WebPagetest Documentation: Speed Index, August 29, 2019. URL: https://web.archive.org/web/20190910/https://sites.google.com/a/webpagetest.org/docs/using-webpagetest/metrics/speed-index (archived on 2019-09-10).

[Goo19c] The Google BBR team. BBR Startup Gain: a Derivation, February 1, 2019. URL: https://github.com/google/bbr/blob/master/Documentation/startup/gain/analysis/bbr_startup_gain.pdf (visited on 2019-07-31).

[Gra19] Robert Graham. MASSCAN: Mass IP port scanner, May 22, 2019. URL: https://web.archive.org/web/20190821/https://github.com/robertdavidgraham/masscan (archived on 2019-08-21).

[Gua17a] The Guardian. Ads don't work so websites are using your electricity to pay the bills, September 27, 2017. URL: https://web.archive.org/web/20180515/https://www.theguardian.com/technology/2017/sep/27/pirate-bay-showtime-ads-websites-electricity-pay-bills-cryptocurrency-bitcoin (archived on 2018-05-15).

[Gua17b] The Guardian. Billions of video site visitors unwittingly mine cryptocurrency as they watch, December 13, 2017. URL: https://web.archive.org/web/20180516/https://www.theguardian.com/technology/2017/dec/13/video-site-visitors-unwittingly-mine-cryptocurrency-as-they-watch-report-openload-streamango-rapidvideo-onlinevideoconverter-monero (archived on 2018-05-16).

[GH18] Hang Guo and John Heidemann. Detecting ICMP Rate Limiting in the Internet. In *Proceedings of the Conference on Passive and Active Measurement (PAM '18)*, pages 3–17. Springer, Cham, 2018. DOI: 10.1007/978-3-319-76481-8_1.

[GOR+19] Dennis Guse, Henrique R. Orefice, Gabriel Reimers, and Oliver Hohlfeld. TheFragebogen: A Web Browser-based Questionnaire Framework for Scientific Research. In *Proceedings of the International Conference on Quality of Multimedia Experience (QoMEX '19)*, pages 1–3. IEEE, 2019. DOI: 10.1109/QoMEX.2019.8743231.

[HR08] Sangtae Ha and Injong Rhee. Hybrid Slow Start for High-Bandwidth and Long-Distance Networks. In *Protocols for Fast, Long Distance Networks Workshop (PFLDnet '08)*, pages 1–6, 2008. URL: https://pdfs.semanticscholar.org/25e9/ef3f03315782c7f1cbcd31b587857adae7d1.pdf (visited on 2019-08-21).

[HRX08] Sangtae Ha, Injong Rhee, and Lisong Xu. CUBIC : A New TCP-Friendly High-Speed TCP Variant. *SIGOPS Operating Systems Review - Research and developments in the Linux kernel (Oper. Syst. Rev. July '08)*, 42(5):64–74, ACM, July 2008. DOI: 10.1145/1400097.1400105.

[Han06] Mark Handley. Why the Internet only just works. *BT Technology Journal (BT Technol J)*, 24(3):119–129, Kluwer Academic Publishers-Consultants Bureau, July 2006. DOI: 10.1007/s10550-006-0084-z.

[HNE+10] Seppo Hätönen, Aki Nyrhinen, Lars Eggert, Stephen Strowes, Pasi Sarolahti, and Markku Kojo. An Experimental Study of Home Gateway Characteristics. In *Proceedings of the Internet Measurement Conference (IMC '10)*, pages 260–266. ACM, 2010. DOI: 10.1145/1879141.1879174.

[HMM+02] Urs Hengartner, Sue Moon, Richard Mortier, and Christophe Diot. Detection and Analysis of Routing Loops in Packet Traces. In *Proceedings of the SIGCOMM Workshop on Internet Measurement (IMW '02)*, pages 107–112. ACM, 2002. DOI: 10.1145/637201.637217.

[Hew10] Hewlett Packard. HP-UX - Serviceguard A.11.19 on HP-UX 11.31: Source Quench Seen for Every IPMON Ping, 2010. URL: https://web.archive.org/web/20190813/https://support.hpe.com/hpsc/doc/public/display?docId=emr_na-c02190964 (archived on 2019-08-13).

[HBZ17] Mario Hock, Roland Bless, and Martina Zitterbart. Experimental Evaluation of BBR Congestion Control. In *Proceedings of the International Conference on Network Protocols (ICNP '17)*, pages 1–10. IEEE, 2017. DOI: 10.1109/ICNP.2017.8117540.

[HPC+14] Oliver Hohlfeld, Enric Pujol, Florin Ciucu, Anja Feldmann, and Paul Barford. A QoE Perspective on Sizing Network Buffers. In *Proceedings of the Internet Measurement Conference (IMC '14)*, pages 333–346. ACM, 2014. DOI: 10.1145/2663716.2663730.

[HRW+18] Oliver Hohlfeld, Jan Rüth, Konrad Wolsing, and Torsten Zimmermann. Characterizing a Meta-CDN. In *Proceedings of the Conference on Passive and Active Measurement (PAM '18)*, pages 114–128. Springer, Cham, 2018. DOI: 10.1007/978-3-319-76481-8_9.

[HLA18] Matthew Holt, Light Code Labs, and Ardan Labs. Caddy - The HTTP/2 Web Server with Automatic HTTPS, 2018. URL: https://web.archive.org/web/20190801/https://caddyserver.com/ (archived on 2019-08-01).

[HHA+20] Ralph Holz, Jens Hiller, Johanna Amann, Abbas Razaghpanah, Thomas Jost, Narseo Vallina-Rodriguez, and Oliver Hohlfeld. Tracking the Deployment of TLS 1.3 on the Web: A Story of Experimentation and Centralization. *SIGCOMM Computer Communication Review (CCR July '20)*, 50(3):3–15, ACM, July 2020. DOI: 10.1145/3411740.3411742.

[Hos18] Hosh (hoshsadiq). Github: Block lists to prevent JavaScript miners, May 17, 2018. URL: https://web.archive.org/web/20180517/https://github.com/hoshsadiq/adblock-nocoin-list (archived on 2018-05-17).

[HKH+14] Tobias Hoßfeld, Christian Keimel, Matthias Hirth, Bruno Gardlo, Julian Habigt, Klaus Diepold, and Phuoc Tran-Gia. Best Practices for QoE Crowdtesting: QoE Assessment With Crowdsourcing. *IEEE Transactions on Multimedia (TMM Feb. '14)*, 16(2):541–558, IEEE, February 2014. DOI: 10.1109/TMM.2013.2291663.

[IET17a] The IETF Community. Interim June 2017 Minutes on: Packet Number Echo PR#269, June 7, 2017. URL: https://web.archive.org/web/20190801/https://github.com/quicwg/wg-materials/blob/master/interim-17-06/minutes.md/ (archived on 2019-08-01).

[IET19] The IETF Community. Applications Doing DNS (add) – Group History, 2019. URL: https://web.archive.org/web/20190918/https://datatracker.ietf.org/wg/add/history/ (archived on 2019-09-18).

[IET17b] The IETF QUIC Working Group. Charter for Working Group, February 27, 2017. URL: https://datatracker.ietf.org/wg/quic/about/ (visited on 2019-07-31).

[ITU03] ITU. Subjective quality evaluation of telephone services based on spoken dialogue systems. ITU-T Recommendation P.851, 2003. URL: https://www.itu.int/rec/T-REC-P.851/en (visited on 2019-09-10).

[Iye15] Jana Iyengar. IETF93 QUIC BarBoF: Protocol Overview, July 22, 2015. URL: https://web.archive.org/web/20190819/https://docs.google.com/presentation/d/15e1bLKYeN56GL1oTJSF9OZiUsI-rcxisLo9dEyDkWQs/edit (archived on 2019-08-19).

[Iye19] Jana Iyengar. QUIC Tutorial @ netdev 0x13, May 20, 2019. URL: https://netdevconf.org/0x13/session.html?tutorial-QUIC (visited on 2019-07-14). Video available at: https://youtu.be/CtsBawwGwns.

[IT19] Jana Iyengar and Martin Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. Internet-Draft draft-ietf-quic-transport-22, Internet Engineering Task Force, July 9, 2019, pages 1–148. URL: https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-22. Work in Progress.

[Jac88] Van Jacobson. Congestion Avoidance and Control. In *Proceedings of the Symposium on Communications Architectures and Protocols (SIGCOMM '88)*, pages 314–329. ACM, 1988. DOI: 10.1145/52324.52356.

[Jac90] Van Jacobson. Modified TCP Congestion Control and Avoidance Algorithms. Mailing List Conversation end2end-interest, April 30, 1990. URL: ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt (visited on 2019-08-19).

[Jac06] Van Jacobson. A Rant on Queues, July 26, 2006. URL: http://www.pollere.net/Pdfdocs/QrantJul06.pdf (visited on 2019-07-18). Talk at MIT Lincoln Labs, Lexington, MA.

[JSS15] Tibor Jager, Jörg Schwenk, and Juraj Somorovsky. On the Security of TLS 1.3 and QUIC Against Weaknesses in PKCS#1 V1.5 Encryption. In *Proceedings of the Conference on Computer and Communications Security (CCS '15)*, pages 1185–1196. ACM, 2015. DOI: 10.1145/2810103.2813657.

[Joh75] Donald B. Johnson. Finding All the Elementary Circuits of a Directed Graph. *Journal on Computing (J. Comput. Mar. '75)*, 4(1):77–84, SIAM, March 1975. DOI: 10.1137/0204007.

[Jun17] Juniper Networks, Inc. no-propagate-ttl - TechLibrary - Juniper Networks, June 27, 2017. URL: https://web.archive.org/web/20190813/https://www.juniper.net/documentation/en_US/junos/topics/reference/configuration-statement/no-propagate-ttl-edit-protocols-mpls.html (archived on 2019-08-13).

[KJC+17] Arash Molavi Kakhki, Samuel Jero, David Choffnes, Alan Mislove, and Cristina Nita-Rotaru. Taking a Long Look at QUIC: An Approach for Rigorous Evaluation of Rapidly Evolving Transport Protocols. In *Proceedings of the Internet Measurement Conference (IMC '17)*, pages 290–303. ACM, 2017. DOI: 10.1145/3131365.3131368.

[KGP+09] Elliott Karpilovsky, Alexandre Gerber, Dan Pei, Jennifer Rexford, and Aman Shaikh. Quantifying the Extent of IPv6 Deployment. In *Proceedings of the Conference on Passive and Active Measurement (PAM '09)*, pages 13–22. Springer Berlin Heidelberg, 2009. DOI: 10.1007/978-3-642-00975-4_2.

[KRB+17] Conor Kelton, Jihoon Ryoo, Aruna Balasubramanian, and Samir R. Das. Improving User Perceived Page Load Times Using Gaze. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI '17)*, pages 545–559. USENIX Association, 2017. URL: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/kelton.

[Kle10] Leonard Kleinrock. An early history of the internet [History of Communications]. *IEEE Communications Magazine (ComMag Aug. '10)*, 48(8):26–36, IEEE, August 2010. DOI: 10.1109/MCOM.2010.5534584.

[KVM+18] Radhesh Krishnan Konoth, Emanuele Vineti, Veelasha Moonsamy, Martina Lindorfer, Christopher Kruegel, Herbert Bos, and Giovanni Vigna. MineSweeper: An In-depth Look into Drive-by Cryptocurrency Mining and Its Defense. In *Proceedings of the Conference on Computer and Communications Security (CCS '18)*, pages 1714–1730. ACM, 2018. DOI: 10.1145/3243734.3243858.

[Kru19] Marijn Kruisselbrink. Native File System. Draft Community Group Report, July 25, 2019. URL: https://web.archive.org/web/20190827061944/https://wicg.github.io/native-file-system/ (archived on 2019-08-27).

[KBT+17] Mirja Kühlewind, Tobias Bühler, Brian Trammell, Stephan Neuhaus, Roman Müntener, and Gorry Fairhurst. A Path Layer for the Internet: Enabling Network Operations on Encrypted Protocols. In *Proceedings of the International Conference on Network and Service Management (CNSM '17)*, pages 1–9. IEEE, 2017. DOI: 10.23919/CNSM.2017.8255973.

[Kun18] Ike Kunze. How Fair Is The Internet? Investigating Bottleneck And Connection Characteristics. Master's Thesis, Rheinisch-Westfälische Technische Hochschule Aachen, September 2018.

[LIM+10] Craig Labovitz, Scott Iekel-Johnson, Danny McPherson, Jon Oberheide, and Farnam Jahanian. Internet Inter-domain Traffic. In *Proceedings of the SIGCOMM Conference (SIGCOMM '10)*, pages 75–86. ACM, 2010. DOI: 10.1145/1851182.1851194.

[LRW+17] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, and Zhongyi Shi. The QUIC Transport Protocol: Design and Internet-Scale Deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*, pages 183–196. ACM, 2017. DOI: 10.1145/3098822.3098842.

[LHM10] Bob Lantz, Brandon Heller, and Nick McKeown. A Network in a Laptop: Rapid Prototyping for Software-defined Networks. In *Proceedings of the SIGCOMM Workshop on Hot Topics in Networks (HOTNETS IX)*, pages 1–19. ACM, 2010. DOI: 10.1145/1868447.1868466.

[LBB+19] Pierre Laperdrix, Nataliia Bielova, Benoit Baudry, and Gildas Avoine. Browser Fingerprinting: A survey. arXiv ePrint 1905.01051, 2019, pages 1–29. URL: https://arxiv.org/abs/1905.01051.

[LCC+09] Barry M. Leiner, Vinton G. Cerf, David D. Clark, Robert E. Kahn, Leonard Kleinrock, Daniel C. Lynch, Jon Postel, Larry G. Roberts, and Stephen Wolff. A Brief History of the Internet. *SIGCOMM Computer Communication Review (CCR Oct. 09)*, 39(5):22–31, ACM, October 2009. DOI: 10.1145/1629607.1629613.

[LSM07] Douglas J. Leith, Robert N. Shorten, and Gavin McCullagh. Experimental evaluation of Cubic-TCP. In *Proceedings of the Workshop on Protocols for Fast Long-Distance Networks (PFLDNet '07)*, pages 1–9, 2007. URL: `https://www.hamilton.ie/net/pfldnet2007_cubic_final.pdf` (visited on 2019-08-21).

[LNC+19] Fangfan Li, Arian Akhavan Niaki, David Choffnes, Phillipa Gill, and Alan Mislove. A Large-scale Analysis of Deployed Traffic Differentiation Practices. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM '19)*, pages 130–144. ACM, 2019. DOI: `10.1145/3341302.3342092`.

[Lin06] Greg Linden. Marissa Mayer at Web 2.0, November 9, 2006. URL: `https://web.archive.org/web/20190603/http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html` (archived on 2019-06-03).

[Lit17] LiteSpeed Technologies Inc. LiteSpeed — Release Log, June 12, 2017. URL: `https://web.archive.org/web/20190731/https://www.litespeedtech.com/products/litespeed-web-server/release-log-archive` (visited on 2019-07-31).

[LAJ+07] Dan Liu, Mark Allman, Shudong Jin, and Limin Wang. Congestion Control Without a Startup Phase. In *Protocols for Fast, Long Distance Networks Workshop (PFLDnet '07)*, pages 61–66, 2007. URL: `https://www.icir.org/mallman/papers/jumpstart-pfldnet07.pdf` (visited on 2019-08-21).

[LWY+12] Hongqiang Harry Liu, Ye Wang, Yang Richard Yang, Hao Wang, and Chen Tian. Optimizing Cost and Performance for Content Multihoming. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '12)*, pages 371–382. ACM, 2012. DOI: `10.1145/2342356.2342432`.

[Löb18] Alexander Löbel. Measuring TCP Initial Windows of Content Delivery Networks. Bachelor's Thesis, Rheinisch-Westfälische Technische Hochschule Aachen, June 2018.

[LLK+17] Qasim Lone, Matthew Luckie, Maciej Korczyński, and Michel van Eeten. Using Loops Observed in Traceroute to Infer the Ability to Spoof. In *Proceedings of the Conference on Passive and Active Measurement (PAM '17)*, pages 229–241. Springer, Cham, 2017. DOI: `10.1007/978-3-319-54328-4_17`.

[LJB+15] Robert Lychev, Samuel Jero, Alexandra Boldyreva, and Christina Nita-Rotaru. How Secure and Quick is QUIC? Provable Security and Performance Analyses. In *Proceedings of the Symposium on Security and Privacy (S&P '15)*, pages 214–231. IEEE, 2015. DOI: `10.1109/SP.2015.21`.

[MJW+17] Shiyao Ma, Jingjie Jiang, Wei Wang, and Bo Li. Fairness of Congestion-Based Congestion Control: Experimental Evaluation and Analysis. arXiv ePrint 1706.09115v2, 2017, pages 1–11. URL: http://arxiv.org/abs/1706.09115v2.

[Mai19] Patrick Maigron. Regional Internet Registries Statistics — World – Autonomous System Number statistics, April 26, 2019. URL: https://web.archive.org/web/20190505/https://www-public.imtbs-tsp.eu/~maigron/RIR_Stats/RIR_Delegations/ByRIR/Stats-ByRIR.html (archived on 2019-05-05).

[ML07] David Malone and Matthew Luckie. Analysis of ICMP Quotations. In *Proceedings of the Conference on Passive and Active Measurement (PAM '07)*, pages 228–232. Springer, Berlin, Heidelberg, 2007. DOI: 10.1007/978-3-540-71617-4_24.

[MAW18] MAWI Working Group Traffic Archive. Packet traces from WIDE backbone, 2018. URL: https://web.archive.org/web/20190604/http://mawi.nezu.wide.ad.jp/mawi/ (archived on 2019-06-04).

[MPF16] Stephen McQuistin, Colin Perkins, and Marwan Fayed. TCP Hollywood: An Unordered, Time-Lined, TCP for Networked Multimedia Applications. In *Proceedings of the IFIP Networking Conference (NETWORKING '16)*, pages 422–430. IFIP, 2016. DOI: 10.1109/IFIPNetworking.2016.7497221.

[MAF05] Alberto Medina, Mark Allman, and Sally Floyd. Measuring the Evolution of Transport Protocols in the Internet. *SIGCOMM Computer Communication Review (CCR Apr. '05)*, 35(2):37–52, ACM, April 2005. DOI: 10.1145/1064413.1064418.

[MKM16] Péter Megyesi, Zsolt Krämer, and Sándor Molnár. How quick is QUIC? In *Proceedings of the International Conference on Communications (ICC '16)*, pages 1–6. IEEE, 2016. DOI: 10.1109/ICC.2016.7510788.

[Mic19] Microworkers.com. The Microworker Platform, 2019. URL: https://web.archive.org/web/20190910/https://www.microworkers.com/ (archived on 2019-09-10).

[MBB08] Dimitrios Miras, Martin Bateman, and Saleem Bhatti. Fairness of High-Speed TCP Stacks. In *Proceedings of the International Conference on Advanced Information Networking and Applications (AINA '08)*, pages 84–92. IEEE, 2008. DOI: 10.1109/AINA.2008.143.

[Mon18] The Monero Project. Monero - secure, private, untraceable, 2018. URL: https://web.archive.org/web/20180517/https://getmonero.org (archived on 2018-05-17).

[Moz18] Mozilla. Internet Health Report – Google dominates browser market, 2018. URL: http://web.archive.org/web/20190917/https://internethealthreport.org/2018/google-dominates-browser-market/ (archived on 2019-09-17).

[MBM+16] Matthew K. Mukerjee, Ilker Nadi Bozkurt, Bruce Maggs, Srinivasan Seshan, and Hui Zhang. The Impact of Brokers on the Future of Content Delivery. In *Proceedings of the SIGCOMM Workshop on Hot Topics in Networks (HotNets '16)*, pages 127–133. ACM, 2016. DOI: 10.1145/3005745.3005749.

[MBR+17] Matthew K. Mukerjee, Ilker Nadi Bozkurt, Devdeep Ray, Bruce M. Maggs, Srinivasan Seshan, and Hui Zhang. Redesigning CDN-Broker Interactions for Improved Content Delivery. In *Proceedings of the International Conference on emerging Networking EXperiments and Technologies (CoNEXT '17)*, pages 68–80. ACM, 2017. DOI: 10.1145/3143361.3143366.

[AFA+19] Mhd Wesam Al-Nabki, Eduardo Fidalgo, Enrique Alegre, and Laura Fernández-Robles. ToRank: Identifying the Most Influential Suspicious Domains in the Tor Network. *Expert Systems with Applications (ESWA June '19)*, 123:212–226, Elsevier, June 2019. DOI: 10.1016/j.eswa.2019.01.029.

[NFL+14] David Naylor, Alessandro Finamore, Ilias Leontiadis, Yan Grunenberger, Marco Mellia, Maurizio Munafò, Konstantina Papagiannaki, and Peter Steenkiste. The Cost of the "S" in HTTPS. In *Proceedings of the International Conference on emerging Networking EXperiments and Technologies (CoNEXT '14)*, pages 133–140. ACM, 2014. DOI: 10.1145/2674005.2674991.

[NdOA+18] Késsia Nepomuceno, Igor Nogueira de Oliveira, Rafael Roque Aschoff, Daniel Bezerra, Maria Silvia Ito, Wesley Melo, Djamel Sadok, and Géza Szabó. QUIC and TCP: A Performance Evaluation. In *Proceedings of the Symposium on Computers and Communications (ISCC '18)*, pages 1–7. IEEE, 2018. DOI: 10.1109/ISCC.2018.8538687.

[Net12] Netflix. Announcing the Netflix Open Connect Network, June 4, 2012. URL: http://web.archive.org/web/20120606/http://blog.netflix.com/2012/06/announcing-netflix-open-connect-network.html (archived on 2012-06-06).

[NSD+15] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: Accurate Record-and-Replay for HTTP. In *Proceedings of the USENIX Annual Technical Conference (ATC '15)*, pages 417–429. USENIX, 2015. URL: https://www.usenix.org/conference/atc15/technical-session/presentation/netravali.

[NJ12] Kathleen Nichols and Van Jacobson. Controlling Queue Delay. *Queue – Networks. Networks (Queue May '12)*, 10(5):20–34, ACM, May 2012. DOI: 10.1145/2208917.2209336.

[NMS+14] Nick Nikiforakis, Federico Maggi, Gianluca Stringhini, M. Zubair Rafique, Wouter Joosen, Christopher Kruegel, Frank Piessens, Giovanni Vigna, and Stefano Zanero. Stranger Danger: Exploring the Ecosystem of Ad-based URL Shortening Services. In *In Proceedings of the World*

Wide Web Conference (WWW '14), pages 51–62. ACM, 2014. DOI: 10.1145/2566486.2567983.

[NS14] Daiyuu Nobori and Yasushi Shinjo. VPN Gate: A Volunteer-Organized Public VPN Relay System with Blocking Resistance for Bypassing Government Censorship Firewalls. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI '14)*, pages 229–241. USENIX, 2014. URL: https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/nobori.

[Nok17] Nokia. Router Configuration Guide Release 15.0.R5. Manual, September 2017. URL: https://infoproducts.alcatel-lucent.com/cgi-bin/dbaccessfilename.cgi/3HE11976AAACTQZZA01_V1_7450%20ESS%207750%20SR%207950%20XRS%20and%20VSR%20Router%20Configuration%20Guide%20R15.0.R5.pdf (visited on 2019-08-13).

[Nor14] William B. Norton. Internet Peering. In *The Internet Peering Playbook*. DrPeering Press, 2014. Chapter 4. URL: http://web.archive.org/web/20190819/http://drpeering.org/HTML_IPP/ipptoc.html (archived on 2019-08-19).

[NSS10] Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun. The Akamai Network: A Platform for High-performance Internet Applications. *SIGOPS Operating Systems Review (Oper. Syst. Rev. July '10)*, 44(3):2–19, ACM, August 2010. DOI: 10.1145/1842733.1842736.

[OM18] Omsk Social Club and !Mediengruppe Bitnik. Cryptorave #5 Alexiety - 0b673cce.xyz, 2018. URL: https://web.archive.org/web/20180515/https://0b673cce.xyz/ (archived on 2018-05-15).

[Ora19] Oracle. Dynamic Steering | DNS Load Balancing | DynDNS | DDNS Trial, 2019. URL: https://web.archive.org/web/20190814/https://dyn.com/dynamic-steering/ (archived on 2019-08-14).

[OSR+12] John S. Otto, Mario A. Sánchez, John P. Rula, and Fabián E. Bustamante. Content Delivery and the Natural Evolution of DNS: Remote DNS Trends, Performance Issues and Alternative Solutions. In *Proceedings of the Internet Measurement Conference (IMC '12)*, pages 523–536. ACM, 2012. DOI: 10.1145/2398776.2398831.

[PF01] Jitendra Padhye and Sally Floyd. On Inferring TCP Behavior. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '01)*, pages 287–298. ACM, 2001. DOI: 10.1145/383059.383083.

[PDB18] Maxime Piraux, Quentin De Coninck, and Olivier Bonaventure. Observing the Evolution of QUIC Implementations. In *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC (EPIQ '18)*, pages 8–14. ACM, 2018. DOI: 10.1145/3284850.3284852.

[Pix17] Pixalate. Pixalate unveils the list of sites secretly mining for cryptocurrency, October 26, 2017. URL: https://web.archive.org/web/20180515/http://blog.pixalate.com/coinhive-cryptocurrency-mining-cpu-site-list (archived on 2018-05-15).

[PFA+10] Ingmar Poese, Benjamin Frank, Bernhard Ager, Georgios Smaragdakis, and Anja Feldmann. Improving Content Delivery Using Provider-aided Distance Information. In *Proceedings of the Internet Measurement Conference (IMC '10)*, pages 22–34, Melbourne, Australia. ACM, 2010. DOI: 10.1145/1879141.1879145.

[Pro18] Proofpoint. Smominru Monero mining botnet making millions for operators, January 31, 2018. URL: https://web.archive.org/web/20180515/https://www.proofpoint.com/us/threat-insight/post/smominru-monero-mining-botnet-making-millions-operators (archived on 2018-05-15).

[Pub19] Public Interest Registry. Zone File Access, 2019. URL: http://www.pir.org/ (visited on 2019-07-31).

[QGM+09] Feng Qian, Alexandre Gerber, Zhuoqing Morley Mao, Subhabrata Sen, Oliver Spatscheck, and Walter Willinger. TCP Revisited: A Fresh Look at TCP in the Wild. In *Proceedings of the Internet Measurement Conference (IMC '09)*, pages 76–89. ACM, 2009. DOI: 10.1145/1644893.1644903.

[RCC+11] Sivasankar Radhakrishnan, Yuchung Cheng, Jerry Chu, Arvind Jain, and Barath Raghavan. TCP Fast Open. In *Proceedings of the International Conference on emerging Networking EXperiments and Technologies (CoNEXT '11)*, pages 1–12. ACM, 2011. DOI: 10.1145/2079296.2079317.

[RPB+12] Costin Raiciu, Christoph Paasch, Sebastien Barre, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI '12)*, pages 399–412. USENIX Association, 2012. URL: https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/raiciu.

[RLK+19] Mohammad Rajiullah, Andra Lutu, Ali Safari Khatouni, Mah-Rukh Fida, Marco Mellia, Anna Brunstrom, Ozgu Alay, Stefan Alfredsson, and Vincenzo Mancuso. Web Experience in Mobile Networks: Lessons from Two Million Page Visits. In *In Proceedings of the World Wide Web Conference (WWW '19)*, pages 1532–1543. ACM, 2019. DOI: 10.1145/3308558.3313606.

[RFC606] L. Peter Deutsch. Host Names On-line. RFC 606, RFC Editor, December 1973, pages 1–3. DOI: 10.17487/RFC0606.

[RFC792] Jonathan B. Postel. Internet Control Message Protocol. RFC 792, RFC Editor, September 1981, pages 1–21. DOI: 10.17487/RFC0792.

[RFC793] Information Sciences Institute. Transmission Control Protocol. RFC 793, RFC Editor, September 1981, pages 1–91. DOI: 10.17487/RFC0793.

[RFC882] Paul Mockapetris. DOMAIN NAMES - CONCEPTS and FACILITIES. RFC 882, RFC Editor, November 1983, pages 1–31. DOI: 10.17487/RFC0882.

[RFC896] John Nagle. Congestion Control in IP/TCP Internetworks. RFC 896, RFC Editor, January 6, 1984, pages 1–9. DOI: 10.17487/RFC0896.

[RFC1122] Robert Braden. Requirements for Internet Hosts - Communication Layers. RFC 1122, RFC Editor, October 1989, pages 1–116. DOI: 10.17487/RFC1122.

[RFC1191] Jeffrey C. Mogul and Steve E. Deering. Path MTU Discovery. RFC 1191, RFC Editor, November 1990, pages 1–19. DOI: 10.17487/RFC1191.

[RFC1700] Joyce K. Reynolds and Jonathan B. Postel. Assigned Numbers. RFC 1700, RFC Editor, October 1994, pages 1–230. DOI: 10.17487/RFC1700.

[RFC1812] Frederick J. Baker. Requirements for IP Version 4 Routers. RFC 1812, RFC Editor, June 1995, pages 1–175. DOI: 10.17487/RFC1812.

[RFC1925] Ross Callon. The Twelve Networking Truths. RFC 1925, RFC Editor, April 1996, pages 1–3. DOI: 10.17487/RFC1925.

[RFC2001] W. Richard Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. RFC 2001, RFC Editor, January 1997, pages 1–6. DOI: 10.17487/RFC2001.

[RFC2414] Mark Allman, Sally Floyd, and Craig Partridge. Increasing TCP's Initial Window. RFC 2414, RFC Editor, September 1998, pages 1–14. DOI: 10.17487/RFC2414.

[RFC2460] Steve E. Deering and Robert M. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460, RFC Editor, December 1998, pages 1–39. DOI: 10.17487/RFC2460.

[RFC3390] Mark Allman, Sally Floyd, and Craig Partridge. Increasing TCP's Initial Window. RFC 3390, RFC Editor, October 2002, pages 1–14. DOI: 10.17487/RFC3390.

[RFC3742] Sally Floyd. Limited Slow-Start for TCP with Large Congestion Windows. RFC 3742, RFC Editor, March 2004, pages 1–7. DOI: 10.17487/RFC3742.

[RFC4271] Yakov Rekhter, Tony Li, and Susan Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271, RFC Editor, January 2006, pages 1–104. DOI: 10.17487/RFC4271.

[RFC5290] Sally Floyd and Mark Allman. Comments on the Usefulness of Simple Best-Effort Traffic. RFC 5290, RFC Editor, July 2009, pages 1–20. DOI: 10.17487/RFC5290.

[RFC5681] Mark Allman, Vern Paxson, and Ethan Blanton. TCP Congestion Control. RFC 5681, RFC Editor, September 2009, pages 1–18. DOI: 10.17487/RFC5681.

[RFC5927] Fernando Gont. ICMP Attacks against TCP. RFC 5927, RFC Editor, July 2010, pages 1–36. DOI: 10.17487/RFC5927.

[RFC6582] Andrei Gurtov, Tom Henderson, Sally Floyd, and Yoshifumi Nishida. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 6582, RFC Editor, April 2012, pages 1–16. DOI: 10.17487/RFC6582.

[RFC6633] Fernando Gont. Deprecation of ICMP Source Quench Messages. RFC 6633, RFC Editor, May 2012, pages 1–8. DOI: 10.17487/RFC6633.

[RFC6928] H.K. Jerry Chu, Nandita Dukkipati, Yuchung Cheng, and Matt Mathis. Increasing TCP's Initial Window. RFC 6928, RFC Editor, April 2013, pages 1–24. DOI: 10.17487/RFC6928.

[RFC7413] Yuchung Cheng, Jerry Chu, Sivasankar Radhakrishnan, and Arvind Jain. TCP Fast Open. RFC 7413, RFC Editor, December 2014, pages 1–26. DOI: 10.17487/RFC7413.

[RFC7540] Mike Belshe, Roberto Peon, and Martin Thomson (Ed.) Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540, RFC Editor, May 2013, pages 1–96. DOI: 10.17487/RFC7540.

[RFC8289] Kathleen Nichols, Van Jacobson, Andrew McGregor, and Jana Iyengar. Controlled Delay Active Queue Management. RFC 8289, RFC Editor, January 2018. DOI: 10.17487/RFC8289.

[RFC8290] Toke Høiland-Jørgensen, Paul McKenney, Dave Taht, Jim Gettys, and Eric Dumazet. The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm. RFC 8290, RFC Editor, January 2018, pages 1–25. DOI: 10.17487/RFC8290.

[RFC8312] Injong Rhee, Lisong Xu, Sangtae Ha, Alexander Zimmermann, Lars Eggert, and Richard Scheffenegger. CUBIC for Fast Long-Distance Networks. RFC 8312, RFC Editor, February 2018, pages 1–18. DOI: 10.17487/RFC8312.

[RFC8446] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, RFC Editor, August 2018, pages 1–160. DOI: 10.17487/RFC8446.

[RFC8484] Paul E. Hoffman and Patrick McManus. DNS Queries over HTTPS (DoH). RFC 8484, RFC Editor, October 2018, pages 1–21. DOI: 10.17487/RFC8484.

[RNB+18] John P. Rula, James Newman, Fabián E. Bustamante, Arash Molavi Kakhki, and David Choffnes. Mile High WiFi: A First Look At In-Flight Internet Connectivity. In *Proceedings of the International Conference on World Wide Web (WWW '18)*, pages 1449–1458. IW3C2, 2018. DOI: 10.1145/3178876.3186057.

[Rüt17] Jan Rüth. ZMap and Modules, August 12, 2017. URL: https://github.com/COMSYS/zmap (visited on 2019-07-30).

[Rüt18a] Jan Rüth. Coinhive Dataset and Tools, September 19, 2018. DOI: 10.5281/zenodo.1421702.

[Rüt18b] Jan Rüth. Coinhive Link Forwarding Example to Youtube, 2018. URL: https://web.archive.org/web/20180516/https://cnhv.co/3w88o (archived on 2018-05-16).

[Rüt18c] Jan Rüth. Github: IW-Prober, May 16, 2018. DOI: 10.5281/zenodo.1247327.

[Rüt18d] Jan Rüth. ICMP Dataset and Tools, 2018. URL: https://web.archive.org/web/20190813/https://icmp.netray.io/ (archived on 2019-08-13).

[Rüt19] Jan Rüth. Active Measurements and Tools, 2019. URL: https://quic.netray.io (visited on 2019-07-31).

[RBH17] Jan Rüth, Christian Bormann, and Oliver Hohlfeld. Large-Scale Scanning of TCP's Initial Window. In *Proceedings of the Internet Measurement Conference (IMC '17)*, pages 304–310. ACM, 2017. DOI: 10.1145/3131365.3131370.

[RH18] Jan Rüth and Oliver Hohlfeld. Demystifying TCP Initial Window Configurations of Content Distribution Networks. In *Proceedings of the IFIP Network Traffic Measurement and Analysis Conference (TMA '18)*, pages 1–8. IEEE, 2018. DOI: 10.23919/TMA.2018.8506549.

[RKH19a] Jan Rüth, Ike Kunze, and Oliver Hohlfeld. An Empirical View on Content Provider Fairness. In *Proceedings of the IFIP Network Traffic Measurement and Analysis Conference (TMA '19)*, pages 1–8. IEEE, 2019. DOI: 10.23919/TMA.2019.8784684.

[RKH19b] Jan Rüth, Ike Kunze, and Oliver Hohlfeld. TCP's Initial Window – Deployment in the Wild and its Impact on Performance. *Transactions on Network and Service Management (TNSM June '19)*, 16(2):389–402, IEEE, June 2019. DOI: 10.1109/TNSM.2019.2896335.

[RPD+18] Jan Rüth, Ingmar Poese, Christoph Dietzel, and Oliver Hohlfeld. A First Look at QUIC in the Wild. In *Proceedings of the Conference on Passive and Active Measurement (PAM '18)*, pages 255–268. Springer, Cham, 2018. DOI: 10.1007/978-3-319-76481-8_19.

[RWS+19] Jan Rüth, Konrad Wolsing, Martin Serror, Klaus Wehrle, and Oliver Hohlfeld. Blitz-starting QUIC Connections. arXiv ePrint 1905.03144, RWTH Aachen University, 2019, pages 1–8.

[RWW+19] Jan Rüth, Konrad Wolsing, Klaus Wehrle, and Oliver Hohlfeld. Perceiving QUIC: Do Users Notice or Even Care? In *Proceedings of the International Conference on emerging Networking EXperiments and Technologies (CoNEXT '19)*, pages 1–7. ACM, 2019. DOI: 10.1145/3359989.3365416.

[RZH19] Jan Rüth, Torsten Zimmermann, and Oliver Hohlfeld. Hidden Treasures — Recycling Large-Scale Internet Measurements to Study the Internet's Control Plane. In *Proceedings of the Conference on Passive and Active Measurement (PAM '19)*, pages 51–67. Springer, Cham, 2019. DOI: 10.1007/978-3-030-15986-3_4.

[RZW+18] Jan Rüth, Torsten Zimmermann, Konrad Wolsing, and Oliver Hohlfeld. Digging into Browser-based Crypto Mining. In *Proceedings of the Internet Measurement Conference (IMC '18)*, pages 70–76. ACM, 2018. DOI: 10.1145/3278532.3278539.

[SRH+19] Constantin Sander, Jan Rüth, Oliver Hohlfeld, and Klaus Wehrle. DeepCCI: Deep Learning-based Passive Congestion Control Identification. In *Proceedings of the SIGCOMM Workshop on Network Meets AI & ML (NetAI '19)*, pages 1–7. ACM, 2019. DOI: 10.1145/3341216.3342211.

[Sch09] Michael Scharf. Performance Evaluation of Fast Startup Congestion Control Schemes. In *Proceedings of the IFIP Networking Conference (NETWORKING '09)*, pages 716–727. Springer Berlin Heidelberg, 2009. DOI: 10.1007/978-3-642-01399-7_56.

[SGS+17] Quirin Scheitle, Oliver Gasser, Patrick Sattler, and Georg Carle. HLOC: Hints-Based Geolocation Leveraging Multiple Measurement Frameworks. In *Proceedings of the IFIP Network Traffic Measurement and Analysis Conference (TMA '17)*, pages 1–9. IEEE, 2017. DOI: 10.23919/TMA.2017.8002903.

[SHG+18] Quirin Scheitle, Oliver Hohlfeld, Julien Gamba, Jonas Jelten, Torsten Zimmermann, Stephen D. Strowes, and Narseo Vallina-Rodriguez. A Long Way to the Top: Significance, Structure, and Stability of Internet Top Lists. In *Proceedings of the Internet Measurement Conference (IMC '18)*, pages 478–493. ACM, 2018. DOI: 10.1145/3278532.3278574.

[SJS+18] Dominik Scholz, Benedikt Jaeger, Lukas Schwaighofer, Daniel Raumer, Fabien Geyer, and Georg Carle. Towards a Deeper Understanding of TCP BBR Congestion Control. In *Proceedings of the IFIP Networking Conference (NETWORKING '18)*, pages 1–9. IEEE, 2018. DOI: 10.23919/IFIPNetworking.2018.8696830.

[Sec17] Paul Sec. Extract from the Top 1M Alexa domains (and also from investigations) using coin-hive mining service, September 28, 2017. URL: https://web.archive.org/web/20180515/https://gist.github.com/PaulSec/029d198a1e049acead74c31db0de1466 (archived on 2018-05-15).

[Seg18] Jérôme Segura. Drive-by cryptomining campaign targets millions of Android users, February 13, 2018. URL: https://web.archive.org/web/20180515/https://blog.malwarebytes.com/threat-analysis/2018/02/drive-by-cryptomining-campaign-attracts-millions-of-android-users/ (archived on 2018-05-15).

[SJN+13] Seigen, Max Jameson, Tuomo Nieminen, Neocortex, and Antonio M. Juarez. CryptoNight Hash Function. CRYPTONOTE STANDARD 008, March 2013, pages 1–9. URL: https://cryptonote.org/cns/cns008.txt.

[SSW+19a] Michael Seufert, Raimund Schatz, Nikolas Wehner, and Pedro Casas. QUICker or not? - an Empirical Analysis of QUIC vs TCP for Video Streaming QoE Provisioning. In *Proceedings of the Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN '19)*, pages 1–6. IEEE, 2019. DOI: 10.1109/ICIN.2019.8685913.

[SSW+19b] Michael Seufert, Raimund Schatz, Nikolas Wehner, Bruno Gardlo, and Pedro Casas. Is QUIC becoming the New TCP? On the Potential Impact of a New Protocol on Networked Multimedia QoE. In *Proceedings of the International Conference on Quality of Multimedia Experience (QoMEX '19)*, pages 1–6. IEEE, 2019. DOI: 10.1109/QoMEX.2019.8743223.

[SV96] Madhavapeddi Shreedhar and George Varghese. Efficient Fair Queuing Using Deficit Round-Robin. *IEEE/ACM Transactions on Networking (ToN June '96)*, 4(3):375–385, IEEE, June 1996. DOI: 10.1109/90.502236.

[sit19] sitespeed.io. Browsertime - Your browser, your page, your scripts!, September 9, 2019. URL: https://web.archive.org/web/20190910/https://github.com/sitespeedio/browsertime (archived on 2019-09-10).

[SGM+19] Steve Souders, Ilya Grigorik, Pat Meenan, and Rick Viscomi. The HTTP Archive, 2019. URL: https://web.archive.org/web/20190731/https://www.httparchive.org/ (archived on 2019-07-31).

[SMD03] Ashwin Sridharan, Sue Moon, and Christophe Diot. On the Correlation between Route Dynamics and Routing Loops. In *Proceedings of the Internet Measurement Conference (IMC '03)*, pages 285–294. ACM, 2003. DOI: 10.1145/948205.948243.

[Sta82] Christopher C. Stacy. Getting Started Computing at the AI Lab. Working Paper 235, MIT Artificial Intelligence Laboratory, September 7, 1982, pages 1–58. URL: https://dspace.mit.edu/bitstream/handle/1721.1/41180/AI_WP_235.pdf (visited on 2019-08-07).

[SCK+09] Ao-Jan Su, David R. Choffnes, Aleksandar Kuzmanovic, and Fabián E. Bustamante. Drafting Behind Akamai: Inferring Network Conditions Based on CDN Redirections. *IEEE/ACM Transactions on Networking (ToN Dec. '09)*, 17(6):1752–1765, IEEE Press, December 2009. DOI: 10.1109/TNET.2009.2022157.

[Sul17] Nick Sullivan. Introducing Zero Round Trip Time Resumption (0-RTT), March 15, 2017. URL: https://web.archive.org/web/20190909/https://blog.cloudflare.com/introducing-0-rtt/ (archived on 2019-09-09).

[Swe16] Ian Swett. QUIC - Deployment Experience @Google, July 20, 2016. URL: https://www.ietf.org/proceedings/96/slides/slides-96-quic-3.pdf (visited on 2019-07-31). Video available at: https://youtu.be/aGvFuvmEufs?t=2490.

[Sym18] Symantec. Advanced Web Intelligence - RuleSpace | Symantec, 2018. URL: https://web.archive.org/web/20180516/https://www.symantec.com/products/rulespace (archived on 2018-05-16).

[Ten18] Tencent Cloud. Tencent Cloud – Industry Intelligence changes the future of the cloud, 2018. URL: https://web.archive.org/web/20190801/https://cloud.tencent.com/ (archived on 2019-08-01).

[Tho19] Martin Thomson. Version-Independent Properties of QUIC. Internet-Draft draft-ietf-quic-invariants-06, Internet Engineering Task Force, July 10, 2019, pages 1–10. URL: https://datatracker.ietf.org/doc/html/draft-ietf-quic-invariants-06. Work in Progress.

[TH14] Brian Trammell and Joe Hildebrand. Evolving Transport in the Internet. *IEEE Internet Computing (IC Sep.-Oct. '14)*, 18(5):60–64, IEEE Computer Society, September 2014. DOI: 10.1109/MIC.2014.91.

[TKB+15] Brian Trammell, Mirja Kühlewind, Damiano Boppart, Iain Learmonth, Gorry Fairhurst, and Richard Scheffenegger. Enabling Internet-Wide Deployment of Explicit Congestion Notification. In *Proceedings of the Conference on Passive and Active Measurement (PAM '15)*, pages 193–205. Springer, Cham, 2015. DOI: 10.1007/978-3-319-15509-8_15.

[Tre18] TrendMicro. Malvertising Campaign Abuses Google's DoubleClick to Deliver Cryptocurrency Miners, January 26, 2018. URL: https://web.archive.org/web/20180515/https://blog.trendmicro.com/trendlabs-security-intelligence/malvertising-campaign-abuses-googles-doubleclick-to-deliver-cryptocurrency-miners/ (archived on 2018-05-15).

[TGD+18] Martino Trevisan, Danilo Giordano, Idilio Drago, Marco Mellia, and Maurizio Munafo. Five Years at the Edge: Watching Internet from the ISP Network. In *Proceedings of the International Conference on emerging Networking EXperiments and Technologies (CoNEXT '18)*, pages 1–12. ACM, 2018. DOI: 10.1145/3281411.3281433.

[VLF+11] Vytautas Valancius, Cristian Lumezanu, Nick Feamster, Ramesh Johari, and Vijay V. Vazirani. How Many Tiers? Pricing in the Internet Transit Market. In *Proceedings of the ACM Conference on Special Interest Group on Data Communication (SIGCOMM '11)*, pages 194–205. ACM, 2011. DOI: 10.1145/2018436.2018459.

[vRJS+16] Roland van Rijswijk-Deij, Mattijs Jonker, Anna Sperotto, and Aiko Pras. A High-Performance, Scalable Infrastructure for Large-Scale Active DNS Measurements. *Journal on Selected Areas in Communications (JSAC June '16)*, 34(6):1877–1888, IEEE, June 2016. DOI: 10.1109/JSAC.2016.2558918.

[VSN+16] Matteo Varvello, Kyle Schomp, David Naylor, Jeremy Blackburn, Alessandro Finamore, and Konstantina Papagiannaki. Is The Web HTTP/2 Yet? In *Proceedings of the Conference on Passive and Active Measurement (PAM '16)*, pages 218–232. Springer, Cham, 2016. DOI: 10.1007/978-3-319-30505-9_17.

[Ver17] Verisign. The Verisign Domain Name Industry Brief, September 2017. URL: https://www.verisign.com/assets/domain-name-report-Q2 2017.pdf (visited on 2019-07-31).

[Ver19] Verisign. Zone Files For Top-Level Domains (TLDs), 2019. URL: https://web.archive.org/web/20190731/https://www.verisign.com/ (archived on 2019-07-31).

[VS94] Curtis Villamizar and Cheng Song. High Performance TCP in ANSNET. *SIGCOMM Computer Communication Review (CCR Oct. 94)*, 24(5):45–60, ACM, October 1994. DOI: 10.1145/205511.205520.

[VH97] Vikram Visweswaraiah and John Heidemann. Improving Restart of Idle TCP Connections. USC TR 97-661, University of Southern California Computer Science Department, November 10, 1997, pages 1–11. URL: https://www.isi.edu/~johnh/PAPERS/Visweswaraiah97b.pdf (visited on 2019-08-21).

[WQG+09] Feng Wang, Jian Qiu, Lixin Gao, and Jia Wang. On Understanding Transient Interdomain Routing Failures. *IEEE/ACM Transactions on Networking (ToN Mar. '09)*, 17(3):740–751, IEEE Press, June 2009. DOI: 10.1109/TNET.2008.2001952.

[War18] Mark Ward. Websites hacked to mint crypto-cash, October 9, 2018. URL: https://web.archive.org/web/20180515/http://www.bbc.com/news/technology-41518351 (archived on 2018-05-15).

[Web18] WebAssembly Community Group. WebAssembly, 2018. URL: https://web.archive.org/web/20180525/https://webassembly.org (archived on 2018-05-25).

[WCL06] David X. Wei, Pei Cao, and Steven H. Low. TCP Pacing Revisited. Unpublished but Available Online, 2006, pages 1–11. URL: http://people.cs.pitt.edu/~ihsan/pacing_cal.pdf (visited on 2019-08-21).

[Whi11] Jason Whitehorn. jsMiner, June 9, 2011. URL: https://web.archive.org/web/20180517/https://github.com/jwhitehorn/jsMiner (archived on 2018-05-17).

[WMQ+18] Maarten Wijnants, Robin Marx, Peter Quax, and Wim Lamotte. HTTP/2 Prioritization and Its Impact on Web Performance. In *In Proceedings of the World Wide Web Conference (WWW '18)*, pages 1755–1764. IW3C2, 2018. DOI: 10.1145/3178876.3186181.

[Wol19] Konrad Wolsing. Does TCP keep up the pace against QUIC and do users even notice? Master's Thesis, RWTH Aachen University, June 2019.

[WRW+19] Konrad Wolsing, Jan Rüth, Klaus Wehrle, and Oliver Hohlfeld. A Performance Perspective on Web Optimized Protocol Stacks: TCP+TLS+HTTP/2 vs. QUIC. In *Proceedings of the Applied Networking Research Workshop (ANRW '19)*, pages 1–7. ACM, 2019. DOI: 10.1145/3340301.3341123.

[XGF05] Jianhong Xia, Lixin Gao, and Teng Fei. Flooding Attacks by Exploiting Persistent Forwarding Loops. In *Proceedings of the Internet Measurement Conference (IMC '05)*, pages 385–390. USENIX Association, 2005. URL: https://www.usenix.org/conference/imc-05/flooding-attacks-exploiting-persistent-forwarding-loops.

[XGF07] Jianhong Xia, Lixin Gao, and Teng Fei. A Measurement Study of Persistent Forwarding Loops on the Internet. *Computer Networks (ComNet Dec. '07)*, 51(17):4780–4796, Elsevier, December 2007. DOI: 10.1016/j.comnet.2007.07.004.

[XCW17] Jing'an Xue, David Choffnes, and Jilong Wang. CDNs Meet CN An Empirical Study of CDN Deployments in China. *IEEE Access (ACCESS Mar. '17)*, 5:5292–5305, IEEE, March 2017. DOI: 10.1109/ACCESS.2017.2682190.

[XKC+14] Lin Xue, Suman Kumar, Cheng Cui, and Seung-Jong Park. A study of fairness among heterogeneous TCP variants over 10 Gbps high-speed optical networks. *Optical Switching and Networking (OSN July '14)*, 13:124–134, Elsevier, July 2014. DOI: 10.1016/j.osn.2014.03.003.

[YLX+11] Peng Yang, Wen Luo, Lisong Xu, Jitender Deogun, and Ying Lu. TCP Congestion Avoidance Algorithm Identification. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS '11)*, pages 310–321. IEEE, 2011. DOI: 10.1109/ICDCS.2011.27.

[YXY17] Yajun Yu, Mingwei Xu, and Yuan Yang. When QUIC meets TCP: An Experimental Study. In *Proceedings of the International Performance Computing and Communications Conference (IPCCC '17)*, pages 1–8. IEEE, 2017. DOI: 10.1109/PCCC.2017.8280429.

[zaf18] zafaco GmbH. Breitbandmessung Ergebnisse als interaktive Darstellung, 2018. URL: https://web.archive.org/web/20181115/https://breitbandmessung.de/interaktive-darstellung (archived on 2018-11-15).

[Zal14] Michał Zalewski. p0f v3, 2014. URL: http://lcamtuf.coredump.cx/p0f3/ (visited on 2019-08-01).

[ZRW+17] Torsten Zimmermann, Jan Rüth, Benedikt Wolters, and Oliver Hohlfeld. How HTTP/2 Pushes the Web: An Empirical Study of HTTP/2 Server Push. In *Proceedings of the IFIP Networking Conference (NETWORKING '17)*, pages 1–9. IEEE, 2017. DOI: 10.23919/IFIPNetworking.2017.8264830.

[ZWH17] Torsten Zimmermann, Benedikt Wolters, and Oliver Hohlfeld. A QoE Perspective on HTTP/2 Server Push. In *Proceedings of the Workshop on QoE-based Analysis and Management of Data Communication Networks (Internet QoE '17)*, pages 1–6. ACM, 2017. DOI: 10.1145/3098603.3098604.

[ZWH+18] Torsten Zimmermann, Benedikt Wolters, Oliver Hohlfeld, and Klaus Wehrle. Is the Web ready for HTTP/2 Server Push? In *Proceedings of the International Conference on emerging Networking EXperiments and Technologies (CoNEXT '18)*, pages 13–19. ACM, 2018. DOI: 10.1145/3281411.3281434.